DIASOFT
всё по-настоящему

Система управления базами данных RuDB

Версия 18-260619

2026

Содержание

1. Общие положения . . . 5
1.1. Что такое RuDB? . . . 5
1.2. Ключевые технологические характеристики . . . 5
2. Установка и настройка СУБД . . . 7
2.1. Совместимые операционные системы . . . 7
2.2. Требования системы . . . 7
2.2.1. Минимальные системные требования . . . 7
2.2.2. Рекомендуемые системные требования . . . 7
2.3. Процесс установки . . . 8
2.3.1. Установка в ОС Astra Linux . . . 9
2.3.2. Установка в ОС Ubuntu . . . 9
2.3.3. Установка в ОС ALT Linux . . . 9
2.3.4. Установка в ОС ОСнова . . . 10
2.4. Результат установки СУБД . . . 10
2.5. Файловая структура . . . 11
2.5.1. Исполняемые файлы . . . 11
2.5.2. Каталог базы данных . . . 12
2.6. Сервисы RuDB . . . 15
2.6.1. Сервис rudb-18 . . . 15
2.6.2. Сервис dgrid . . . 16
3. Определение версии RuDB . . . 17
4. Модуль TRANDLOG: логирование и подмена SQL-запросов . . . 18
4.1. Назначение . . . 18
4.2. Подготовка окружения . . . 18
4.3. Принцип работы . . . 18
4.4. Формат файла trandlog.csv . . . 19
4.5. Параметры конфигурации . . . 19
4.6. Настройка режима работы . . . 21
4.7. Формат логов . . . 21
4.8. Пример сценария работы . . . 22
4.9. Особенности и ограничения . . . 22

5. Встроенные пакеты, функции и процедуры . . . 24
5.1. Общие сведения . . . 24
5.2. Описание объектов пакетов . . . 25
5.2.1. Пакет DBMS_OUTPUT . . . 25
5.2.2. Пакет DBMS_LOB . . . 34
5.2.3. Пакет DBMS_ASSERT . . . 42
5.2.4. Пакет DBMS_RANDOM . . . 46
5.2.5. Пакет DBMS_AQADM . . . 53
5.2.6. Пакет DBMS_ALERT . . . 63
5.2.7. Пакет DBMS_DESCRIBE . . . 70
5.2.8. Пакет DBMS_REPORT . . . 73
5.2.9. Пакет DBMS_UTILITY . . . 76
5.2.10. Пакет DBMS_XMLGEN . . . 82
5.2.11. Пакет DBMS_CRYPTO . . . 98
5.2.12. Пакет DBMS_HASH . . . 101
5.2.13. Пакет DBMS_ODCI . . . 103
5.2.14. Пакет OWA . . . 106
5.2.15. Пакет OWA_COOKIE . . . 107
5.2.16. Пакет OWA_CACHE . . . 108
5.2.17. Пакет OWA_UTIL . . . 117
5.2.18. Пакет DBMS_JSON . . . 119
5.2.19. Пакет DBMS_JSON_SCHEMA . . . 123
5.2.20. Пакет DBMS_FILE_TRANSFER . . . 126
5.2.21. Пакет DBMS_SYSTEM . . . 133
5.2.22. Пакет DBMS_TNS . . . 137
5.2.23. Пакет DBMS_WARNING . . . 139
5.2.24. Пакет DBMS_TRACE . . . 144
5.2.25. Пакет DBMS_DB_VERSION . . . 150
5.2.26. Пакет DBMS_STAT_FUNCS . . . 159
5.2.27. Пакет DBMS_SQLSET . . . 166
5.2.28. Пакет DBMS_DDL . . . 184
5.2.29. Пакет DBMS_XMLSAVE . . . 185
5.2.30. Пакет DBMS_XMLSTORE . . . 187

5.2.31. Пакет OWA_MATCH . . . 192
5.2.32. Пакет OWA_PATTERN . . . 193
5.2.33. Пакет OWA_TEXT . . . 197
5.2.34. Пакет OWA_UTIL . . . 201
5.2.35. Пакет DBMS_HTP . . . 206
5.2.36. Пакет DBMS_HTF . . . 246
5.2.37. Пакет UTL_ENCODE . . . 279
5.2.38. Пакет UTL_INNADR . . . 285
5.2.39. Пакет UTL_MATCH . . . 288
5.2.40. Пакет UTL_RAW . . . 291
5.2.41. Пакет UTL_SMTP . . . 309
5.2.42. Пакет UTL_TCP . . . 332
5.2.43. Пакет UTL_URL . . . 347
5.2.44. Пакет UTL_CALL_STACK . . . 349
5.2.45. Пакет UTL_HTTP . . . 355
5.2.46. Пакет UTL_COMPRESS . . . 393
5.2.47. Пакет UTL_FILE . . . 403
5.2.48. Пакет UTL_I18N . . . 419
5.2.49. Пакет UTL_MAIL . . . 431
5.2.50. Пакет UTL_REF . . . 436
5.2.51. Пакет UTL_GDK . . . 442
5.2.52. Пакет PLVDATE . . . 446
5.2.53. Пакет PLVSTR . . . 471
5.2.54. Пакет PLVCHR . . . 483
6. In-memory-компонента . . . 486

1. Общие положения

1.1. Что такое RuDB?

RuDB — это российская независимая сборка СУБД на основе PostgreSQL, предназначенная для построения отказоустойчивых, масштабируемых и безопасных решений в государственном и корпоративном секторах. СУБД сочетает в себе строгую транзакционность (полное соответствие принципам ACID) и гибкость объектно-реляционной модели, обеспечивая эффективную обработку как традиционных, так и современных типов данных.

Поддерживаемые операционные системы

RuDB доступна для установки и эксплуатации на следующих операционных системах:

- ALT Linux
- Astra Linux
- Ubuntu
- ОСнова

1.2. Ключевые технологические характеристики

1. **Архитектура:** многопроцессная модель (отдельный процесс на каждое соединение), что обеспечивает высокую изоляцию и стабильность.
2. **Стандарты SQL:** реализована значительная часть стандарта SQL:2016 (включая оконные функции, рекурсивные WITH, MERGE).
3. **ACID и многоверсионность:** строгая транзакционность через механизм MVCC и журнал предзаписи (WAL) гарантирует целостность данных даже при сбоях.
4. **Типы данных:** помимо стандартных (NUMERIC, VARCHAR, JSON/JSONB, XML, UUID), поддерживаются геометрические, составные типы, перечисления и битовые строки.
5. **Индексация:** B-tree, Hash, GiST, GIN, BRIN, включая частичные и функциональные индексы.
6. **Расширяемость:** возможность добавления новых типов данных, операторов и функций на C/C++ без изменения ядра.

СУБД RuDB поддерживает базовые функциональные возможности PostgreSQL, а также обеспечивает:

- **Безопасность:** аутентификация через SCRAM-SHA-256, GSSAPI, LDAP, PAM; поддержка SSL/TLS; ролевая модель с наследованием привилегий и разделением по схемам.
- **Отказоустойчивость и репликация:**
  - синхронная и асинхронная потоковая репликация;
  - логическая репликация на уровне таблиц;
  - кластеризация с использованием etcd/Patroni для автоматического failover.
- **Управление планом выполнения запросов** с помощью указаний, записываемых в виде простых описаний.

Возможности СУБД RuDB:

- **In-memory обработка** — распределенное key-value хранилище для высокоскоростных операций.
- **Поколоночное хранение** — оптимизация для аналитических запросов и больших объемов данных.
- **Векторный поиск** — поддержка AI/ML-задач и поиска по семантическим признакам.
- **Мониторинг:** системные представления `pg_stat_*` и динамический сбор статистики запросов.

2. Установка и настройка СУБД

2.1. Совместимые операционные системы

Дистрибутивы выпускается для следующих ОС:

- ALT Linux – версия 11
- Astra Linux – версия 1.8
- Ubuntu – версия 24
- ОСнова - версия 3.1 onyx

2.2. Требования системы

2.2.1. Минимальные системные требования

Процессор

- Многопоточность/ядра CPU: минимум 2 ядра, лучше — 4+ ядер.
- Тактовая частота процессора: от 2 ГГц и выше.

Оперативная память (RAM)

- Минимум: 8 ГБ RAM.
- Оптимальное значение: от 16 ГБ RAM и выше.

Хранилище (HDD/SDD)

- Дисковая подсистема должна поддерживать высокую производительность ввода-вывода (I/*).
- Рекомендуемый объем хранилища зависит от объема данных и рабочих нагрузок, но минимально около 50–100 ГБ свободного пространства.
- Для больших объемов данных желательно использование SSD-дисков или RAID-массивов для повышения производительности операций чтения-записи.

2.2.2. Рекомендуемые системные требования

Процессор

- Количество ядер: 8+ (желательно больше, особенно для масштабируемых решений).
- Тактовая частота процессора: от 2,5 ГГц и выше.
- Гиперпоточность (Hyper-Threading), поддержка AVX-инструкций и технологии виртуализации (VT-x / AMD-V).

Оперативная память (RAM)

- Минимум: 32 ГБ RAM.
- Желательно: 64 ГБ и выше, в зависимости от размера базы данных и сложности приложений.

Хранилище (SSD/HDD)

- Быстрая дисковая подсистема с высокой пропускной способностью IOPS (операций ввода-вывода в секунду). Рекомендуется NVMe или быстрый SSD.
- Объем диска должен соответствовать размеру баз данных плюс запас на будущее расширение и хранение временных файлов (лог-файлы, транзакционные журналы, временные файлы сортировки и индексов).
- Использовать аппаратные RAID-конфигурации уровня 10 или программные аналоги для увеличения отказоустойчивости и скорости записи/чтения.

2.3. Процесс установки

Предварительно перед установкой необходимо проверить, что порт 5432 не занят на сервере, на котором производится установка.

Проверка осуществляется с помощью команды (для всех дистрибутивов ОС Linux):

```
sudo netstat -tulpn
```

Если порт занят, возникнут ошибки при установке СУБД.

В случае невозможности освободить данный порт – после неудачной установки необходимо сменить используемый порт СУБД в файле postgresql.conf и перезапустить все сервисы СУБД.

2.3.1. Установка в ОС Astra Linux

Шаги установки (удаления) пакета:

1. Подключить стандартный репозиторий пакетов текущей ОС.
2. Загрузить необходимую версию пакета СУБД RuDB.
3. Установить пакет:

```
sudo apt install ./rudb-18.26042001-astra1.8_x86_64.deb;
```

При необходимости удаления использовать команду:

```
sudo apt remove rudb-18;
```

2.3.2. Установка в ОС Ubuntu

Шаги установки (удаления) пакета:

1. Подключить стандартный репозиторий пакетов текущей ОС.
2. Загрузить необходимую версию пакета СУБД RuDB.
3. Установить пакет:

```
sudo apt install ./rudb-18.26042001-ubuntu24_x86_64.deb;
```

При необходимости удаления использовать команду:

```
sudo apt remove rudb-18;
```

2.3.3. Установка в ОС ALT Linux

Шаги установки (удаления) пакета:

1. Подключить стандартный репозиторий пакетов текущей ОС.
2. Загрузить необходимую версию пакета СУБД RuDB.
3. Установить пакет:

```
apt-get install ./rudb-18.26042001-alt11_x86_64.rpm;
При необходимости удаления использовать команду:
[source, console]
[%unbreakable]
apt-get remove rudb-18;
```

Внимание! Каталог с базами данных в этой ОС не будет удален ни при удалении, ни при переустановке rpm-пакета. Если необходима полная переустановка, выполните удаление каталога данных вручную.

2.3.4. Установка в ОС ОСнова

Шаги установки (удаления) пакета:

1. Подключить стандартный репозиторий пакетов текущей ОС.
2. Загрузить необходимую версию пакета СУБД RuDB.
3. Установить пакет:

```
sudo apt install ./rudb-18.26042001-onyx3.1_x86_64.deb;
```

При необходимости удаления использовать команду:

```
sudo apt remove rudb-18;
```

2.4. Результат установки СУБД

После успешной установки сервер RuDB запускается и становится доступен на порте 5432.

Внимание! Данный порт не будет доступен, если подключение к СУБД происходит извне! Для удаленного (не локального) взаимодействия любого приложения с СУБД требуется дополнить список разрешенных IP в файле pg_hba.conf. Подробнее об этом файле читайте далее в следующем подразделе "Файловая структура".

Все исполняемые модули размещены в каталоге:

/opt/rudb-18

Базы данных размещены в каталоге:

/var/lib/rudb-18/data

В фоне начинают работу 2 сервиса: rudb-18, dgrid. По умолчанию создается БД:

- rudb - суперпользователь "postgres" с паролем "postgres";

2.5. Файловая структура

2.5.1. Исполняемые файлы

Актуально для всех ОС*

Все исполняемые модули размещены в каталоге:

/opt/rudb-18/bin

В данном каталоге расположены файлы, упрощающие работу с диалектом PostgreSQL:

1. **postgres**: Основной серверный процесс БД. Он отвечает за обработку запросов и управление базой данных.
2. **pg_ctl**: Утилита для управления сервером БД. Позволяет запускать, останавливать и перезапускать сервер.
3. **initdb**: Утилита для инициализации новой базы данных. Создает кластер баз данных и настраивает его.
4. **createdb**: Утилита для создания новой базы данных в кластере.
5. **createuser**: Утилита для создания нового пользователя (роли).
6. **dropdb**: Утилита для удаления базы данных из кластера.
7. **dropuser**: Утилита для удаления пользователя (роли).
8. **pg_dump**: Утилита для создания резервной копии базы данных.
9. **pg_restore**: Утилита для восстановления базы данных из резервной копии, созданной с помощью pg_dump.
10. **pg_basebackup**: Утилита для создания полной резервной копии кластера.

2.5.2. Каталог базы данных

Все базы данных размещены в каталоге:

/var/lib/rudb-18/data

Данный каталог содержит конфигурационные файлы. Среди них присутствуют два важных файла:

1. postgresql.conf
2. pg_hba.conf

Рассмотрим их назначение.

Файл postgresql.conf

Это главный файл конфигурации БД, содержащий глобальные настройки, влияющие на работу всей системы баз данных. Здесь определяются ключевые параметры производительности, памяти, журналирования, репликации и другие аспекты функционирования БД, эмулирующей PostgreSQL.

Структура файла

Файл организован в виде секций, каждая из которых охватывает определённую группу параметров:

- Общие настройки
- Параметры файловой системы
- Журналирование
- Производительность и память
- Репликация и синхронизация
- Безопасность и контроль доступа

Параметры

Применение изменений

Изменённые параметры вступают в силу после перезагрузки основного сервиса СУБД:

```
sudo systemctl restart rudb-18
```

Или с помощью SQL-команды:

```
SELECT pg_reload_conf();
```

Некоторые параметры требуют полной перезагрузки службы, что указывается комментарием рядом с параметром в файле конфигурации.

Файл pg_hba.conf

Это главный файл конфигурации pg_hba.conf используется в СУБД для определения правил аутентификации клиентов. Эти правила задают, каким пользователям разрешено подключаться к базе данных, откуда они могут подключаться и какой метод аутентификации использовать. Формат файла:

TYPE	DATABASE	USER	ADDRESS	METHOD
local	all	all		trust
local	all	all		md5
host	all	postgres	127.0.0.1/32	trust
host	all	all	127.0.0.1/32	md5

Каждая строка файла соответствует одному правилу аутентификации.

Описание полей:

- **Type** — определяет тип подключения:
- **local**: локальные подключения через Unix сокеты.
- **host**: TCP/IP подключения с шифрованием или без.
- **hostssl**: требует SSL-шифрования для подключения.
- **hostnossl**: запрещает использование SSL-шифрования.
- **Database** — база данных, к которой применяется правило. Можно указывать конкретные имена баз данных, группы баз данных (обозначаются звёздочкой *) или слово all для всех баз данных.
- **User** — пользователь, к которому применяется данное правило. Аналогично

предыдущему полю, допустимы индивидуальные имена пользователей, группы пользователей (со звёздочками) или all для всех пользователей.

- **Address** — адрес компьютера или диапазон IP-адресов, которым разрешен доступ. Может задаваться в формате CIDR (например, 192.168.1.0/24), IPv6-адрес или отдельное значение для конкретного IP-адреса.
- **Method** — способ аутентификации:
- **trust**: полное доверие без проверки пароля.
- **md5**: проверка MD5-хэша пароля.
- **password**: проверка открытого текста пароля.
- **scram-sha-256**: современный метод аутентификации с безопасным алгоритмом SHA-256. Примечание: данный способ недоступен при работе через протокол TDS.
- **ident**: идентификация пользователя по системе ОС.
- **ldap**: аутентификация через LDAP-сервер.
- **cert**: аутентификация сертификатом.
- **Options** — необязательные параметры, такие как срок жизни сеанса или информация о LDAP-сервере.

Примеры записей в файле pg_hba.conf:

1. Разрешить локальному пользователю postgres подключение ко всем базам данных без пароля

```
local all postgres trust
```

2. Любые пользователи с IP 192.168.1.* могут подключаться к базе данных mydb используя md5-аутентификацию

```
host mydb all 192.168.1.0/24 md5
```

3. Только пользователь admin может подключаться к базе datasecure с зашифрованным каналом и методом scram-sha-256

```
host mydb all 192
hostssl datasecure admin 172.16.0.0/16 scram-sha-256
```

4. Все пользователи могут подключаться к любым базам данных без шифрования, используя стандартный пароль

```
host mydb all 192
host all all 0.0.0.0/0 password
```

Порядок применения правил:

Правила применяются последовательно сверху вниз. Первое подходящее правило для текущего подключения считается применённым. Поэтому важно располагать более строгие ограничения выше менее строгих.

Изменение настроек:

После изменения файла pg_hba.conf необходимо перезагрузить конфигурацию БД, эмулирующей PostgreSQL, командой:

```
sudo systemctl reload rudb-18
```

или вручную через SQL-команду:

```
SELECT pg_reload_conf();
```

2.6. Сервисы RuDB

2.6.1. Сервис rudb-18

rudb-18 – это основной сервис СУБД, обеспечивающий работоспособность всех её функциональных возможностей. Данный сервис является ядром СУБД.

Ядро СУБД представляет собой центральный компонент, обеспечивающий выполнение основных функций обработки данных и управление ими. Оно отвечает за обработку запросов пользователей, обеспечение целостности данных, оптимизацию производительности и поддержку параллельной работы множества клиентов одновременно.

2.6.2. Сервис dgrid

dgrid — это сервис встроенной in-memory-компоненты DGrid, предназначенной для организации хранения данных в оперативной памяти внутри процесса СУБД. Компонента предоставляет механизмы для оперативного добавления, выборки и проверки записей, используя память как основное хранилище. Это решение предназначено для использования в качестве высокопроизводительного кэша или буфера для рабочих данных внутри базы данных, что позволяет ускорить обработку часто используемой информации и снизить нагрузку на основную табличную структуру. Подробнее о работе с компонентой читайте далее в отдельном разделе.

3. Определение версии RuDB

В RuDB существуют две функции для вывода текущей версии продукта.

Функция version

Функция version() полностью повторяет поведение оригинальной функции из PostgreSQL и содержит подробную информацию о версии PostgreSQL, включая номер версии и дополнительные детали о сборке.

Пример использования:

```
postgres=# SELECT version();
version
-----------------------------------------------------------------------------------------
-----------------
PostgreSQL 18.2 on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 13.3.0-6ubuntu2~24.04)
13.3.0, 64-bit
(1 row)
```

Функция rudb_version

Функция rudb_version() выводит наименование и версию продукта, а также номер сборки, информацию о лицензии и дополнительные детали о сборке.

Пример использования:

```
postgres=# SELECT rudb_version();
rudb_version
-----------------------------------------------------------------------------------------
---------------
RuDB Database Management System, version 26.04, build 20260420
(1 row)
```

4. Модуль TRANDLOG: логирование и подмена SQL-запросов

4.1. Назначение

- Регистрация всех входящих SQL-запросов
- Формирование шаблонов (масок) запросов
- Поиск и применение правил подмены
- Гибкое логирование с учетом производительности
- Привязка логов к идентификатору соединения (session/connection id)

4.2. Подготовка окружения

Перед использованием необходимо создать файлы и задать права доступа:

```
mkdir -p /opt/rudb-18/logs
mkdir -p /opt/rudb-18/config
touch /opt/rudb-18/config/trandlog.csv
touch /opt/rudb-18/logs/trandlog.log
chown postgres:postgres /opt/rudb-18/config/trandlog.csv
chown postgres:postgres /opt/rudb-18/logs/trandlog.log
chmod 600 /opt/rudb-18/config/trandlog.csv
chmod 600 /opt/rudb-18/logs/trandlog.log
```

Файлы:

- `/opt/rudb-18/config/trandlog.csv` — таблица шаблонов и подмен
- `/opt/rudb-18/logs/trandlog.log` — лог выполнения

4.3. Принцип работы

Каждый SQL-запрос перехватывается, из запроса формируется **маска** (обобщенный шаблон). Для маски вычисляется **SHA1-хэш**, который ищется в `trandlog.csv`.

1. Возможны варианты:
   - Запись не найдена → маска добавляется в файл (`trandlog.csv`)
   - Найдена, но без указанной подмены → запрос выполняется без

изменений

- Найдена с подменой → выполняется модифицированный SQL

2. При включенном логировании записывается информация о выполнении

4.4. Формат файла trandlog.csv

Строка имеет формат:

```
HASH|INPUT_MASK|OUTPUT_MASK|F1|F2|F3
```

где:

- `HASH` — SHA1 от `INPUT_MASK`
- `INPUT_MASK` — шаблон входного запроса
- `OUTPUT_MASK` — шаблон подмены
- `F1..F3` — зарезервированные поля

Пример:

```
cb92b0e4f225980da3d072cb7448a31872147b28|SELECT %$SQLPARAM1$% ;|SELECT %$SQLPARAM1$%
FROM DUAL;||
```

4.5. Параметры конфигурации

Настройка осуществляется через GUC-параметры PostgreSQL.

trandlog.no_conversion

- **Тип:** `boolean`
- **Значения:** `on` / `off`
- **Описание:** отключает механизм подмены SQL
- **По умолчанию:** `on`

trandlog.full_conversion

- **Тип:** `boolean`
- **Значения:** `on` / `off`

- **Описание:** включает полный режим подмены (поиск и применение шаблонов)
- **По умолчанию:** `on`

trandlog.no_logging

- **Тип:** `boolean`
- **Значения:** `on` / `off`
- **Описание:** отключает логирование
- **По умолчанию:** `on`

trandlog.id_len

- **Тип:** `integer`
- **Диапазон:** `0`–`8`
- **Описание:** количество байт идентификатора соединения в логе
- **По умолчанию:** `4`

Особенности:

- Выводимое значение id обрезается до указанного количества байт
- Отображается в hex-формате

trandlog.log_format

- **Тип:** `integer` (битовая маска)
- **Диапазон:** `1`–`7`

Определяет формат логируемой строки:

Значение	Описание
1	Только входной запрос (mask)
2	Только итоговый запрос (output)
3	input ← output
4	Меняет направление стрелки
5	output → input

Значение	Описание
6	только output (альтернативный режим)
7	output → input (полный режим)

4.6. Настройка режима работы

Рекомендуемая конфигурация:

```
ALTER ROLE postgres SET trandlog.no_conversion TO 'off';
ALTER ROLE postgres SET trandlog.full_conversion TO 'on';
ALTER ROLE postgres SET trandlog.no_logging TO 'on';
```

Включение логирования:

```
ALTER ROLE postgres SET trandlog.no_logging TO 'off';
```

Применение настроек:

- Перезапустить сервер PostgreSQL

Проверка параметров:

```
SHOW trandlog.no_conversion;
SHOW trandlog.id_len;
SHOW trandlog.log_format;
```

4.7. Формат логов

Лог записывается в файл:

`/opt/rudb-18/logs/trandlog.log`

Формат строки:

```
[connection_id] timestamp duration_us query_info
```

Пример:

```
[1A2B3C4D] 12:34:56.123456 150.321 SELECT * FROM dual
```

где:

- `connection_id` — идентификатор соединения (обрезанный)
- `timestamp` — время начала выполнения
- `duration_us` — длительность в микросекундах
- `query_info` — строка в зависимости от `log_format`

4.8. Пример сценария работы

Шаг 1. Первый запуск запроса

```
SELECT 1;
```

В `trandlog.csv` добавится:

```
hash|SELECT %$SQLPARAM1$% ;|||
```

Шаг 2. Добавление подмены

```
hash|SELECT %$SQLPARAM1$% ;|SELECT %$SQLPARAM1$% FROM DUAL;||
```

Шаг 3. Повторный запрос

На входе:

```
SELECT 1;
```

На исполнении:

```
SELECT 1 FROM DUAL;
```

4.9. Особенности и ограничения

- Не перехватывает extended protocol (prepared statements)

- Максимальная длина маски ограничена (10240 символов)

5. Встроенные пакеты, функции и процедуры

5.1. Общие сведения

СУБД содержит определенный набор процедур, функций, аналогичных функционалу пакетов зарубежных СУБД.

DBMS-пакеты — это специализированные программные компоненты, предназначенные для облегчения управления базами данных и расширения функциональности СУБД. Каждый пакет включает набор процедур, функций и переменных, предназначенных для выполнения определённых задач, таких как управление производительностью, обеспечение безопасности, ведение журнала аудита и другие.

Зачем нужны DBMS-пакеты?

- **упрощение администрирования:** упрощают выполнение повседневных задач администрирования баз данных;
- **автоматизация задач:** позволяют автоматизировать регулярные операции, такие как резервное копирование, очистка журнала аудита и т.д;
- **расширение функциональности:** повышают возможности СУБД, предоставляя новые инструменты и возможности.

UTL-пакеты (Utility Packages) — это набор предопределённых пакетов, предоставляющих служебные функции для взаимодействия с операционной системой, сетью, файлами и для выполнения других административных задач, которые невозможно реализовать средствами чистого SQL.

Зачем нужны UTL-пакеты?

- **работа с внешними данными**: позволяют читать и записывать текстовые файлы прямо из БД;
- **сетевое взаимодействие**: обеспечивают возможность отправки HTTP-запросов и работы с сетевыми данными;
- **встроенная почта**: дают возможность отправлять электронные письма непосредственно из СУБД;
- **сервисные операции**: предоставляют функции для форматирования данных, работы с кодами ошибок и другие утилиты.

PLVSTR (PL/Vision STRing) — это пакет из библиотеки PL/Vision, предоставляющий расширенный набор утилит для работы со строками, которые дополняют стандартные функции.

Зачем нужен PLVSTR?

- **расширенный анализ строк**: предоставляет функции для поиска подстрок, подсчёта вхождений и более сложного анализа.
- **улучшенная обработка**: позволяет легко разбивать строки на части, удалять лишние символы и выполнять сложные манипуляции.
- **упрощение кода**: инкапсулирует сложную логику работы со строками в простые для вызова процедуры и функции.

PLVDATE (PL/Vision DATE) — это пакет из библиотеки PL/Vision, который расширяет возможности стандартной работы с датами.

Зачем нужен PLVDATE?

- **расширенная арифметика дат**: упрощает вычисления с датами (например, "последний день предыдущего месяца").
- **интуитивные проверки**: предоставляет функции для легкой проверки, является ли дата рабочим днём, и для форматирования вывода.
- **упрощение временных интервалов**: предлагает удобные методы для работы с временными промежутками и календарём.

Поддерживаемые процедуры, функции зарубежных пакетов*:

5.2. Описание объектов пакетов

5.2.1. Пакет DBMS_OUTPUT

DBMS_OUTPUT_PUT

Процедура отправляет часть строки в буфер вывода без добавления символа перевода строки. Текст накапливается в буфере до тех пор, пока не будет вызвана процедура PUT_LINE или NEW_LINE.

Входящие параметры:

- item (VARCHAR2) – строка, которую нужно вывести в буфер.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Постепенное формирование длинных строк вывода несколькими вызовами.
- Вывод текста без переноса строки для последующего дополнения.
- Создание пользовательских форматов вывода с разделением элементов.

Примеры запросов:

```
DO $$
DECLARE
    v_line VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT_ENABLE();
    PERFORM DBMS_OUTPUT_PUT('Hello');
    PERFORM DBMS_OUTPUT_PUT(' World');
    PERFORM DBMS_OUTPUT_NEW_LINE();

    SELECT line, status INTO v_line, v_status FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);

    RAISE NOTICE 'Прочитано: line="%", status=%', v_line, v_status;

    IF v_line = 'Hello World' THEN
        RAISE NOTICE '✅ PUT правильно собрал строку';
    ELSE
        RAISE EXCEPTION '❌ Ошибка PUT';
    END IF;

    RAISE NOTICE 'DBMS_OUTPUT_PUT: УСПЕШНО';
END;
$$;
```

DBMS_OUTPUT_PUT_LINE

Функция отправляет строку в буфер вывода, откуда она может быть отображена средствами SQL*Plus или аналогичными инструментами.

Входящие параметры:

- item (VARCHAR2) – строка, которую нужно вывести в буфер.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Отладка хранимых процедур и триггеров путём вывода промежуточных сообщений.
- Вывод итоговых результатов запросов или вычислений.
- Сообщение ошибок и предупреждений разработчику.

Примеры запросов:

```
DO $$
DECLARE
    v_line VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT_ENABLE();
    PERFORM DBMS_OUTPUT_PUT_LINE('Hello World');

    SELECT line, status INTO v_line, v_status FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);

    RAISE NOTICE 'Прочитано: line="%", status=%', v_line, v_status;

    IF v_line = 'Hello World' THEN
        RAISE NOTICE '□ PUT_LINE записал строку';
    ELSE
        RAISE EXCEPTION '□ Ошибка';
    END IF;

    RAISE NOTICE 'DBMS_OUTPUT_PUT_LINE: УСПЕШНО';
END;
$$;
```

DBMS_OUTPUT_DISABLE

Функция отключает вывод сообщений в буфер и очищает буфер от накопленных данных. После отключения любые последующие вызовы процедур PUT, PUT_LINE и аналогичных игнорируются до повторного включения вывода командой ENABLE.

Входящие параметры:

- Нет параметров.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Завершение сеанса вывода отладочных сообщений после завершения тестирования процедуры.
- Очистка буфера вывода, чтобы избежать ненужных данных в последующих операциях.

Примеры запросов:

```
DO $$
DECLARE
    v_line VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT_ENABLE();
    PERFORM DBMS_OUTPUT_PUT_LINE('before_disable');
    PERFORM DBMS_OUTPUT_DISABLE();

    SELECT line, status INTO v_line, v_status FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);

    RAISE NOTICE 'Прочитано после DISABLE: line="%", status=%', v_line, v_status;

    IF v_status = 1 AND v_line IS NULL THEN
        RAISE NOTICE '□ Буфер отключен, сообщение не прочитано';
    ELSE
        RAISE NOTICE '□□ Буфер возможно активен';
    END IF;

    RAISE NOTICE 'DBMS_OUTPUT_DISABLE: УСПЕШНО';
END;
$$;
```

DBMS_OUTPUT_GET_LINE

Функция получает одну строку из буфера вывода, куда были отправлены сообщения процедурами PUT и PUT_LINE. Эта функция полезна для построчного считывания информации из буфера и последующего её отображения.

Входящие параметры:

- line (VARCHAR2) – переменная, в которой сохраняется извлечённая строка из буфера.
- status (NUMBER) – статус выполнения:
- 0 — операция выполнена успешно.
- 1 — в буфере больше нет строк для извлечения.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Построчное чтение отладочной информации из буфера после выполнения сложной хранимой процедуры.
- Извлечение результатов запросов или логирования для анализа или документирования.

Примеры запросов:

```
DO $$
DECLARE
    v_line1 VARCHAR2;
    v_line2 VARCHAR2;
    v_line3 VARCHAR2;
    v_status1 INTEGER;
    v_status2 INTEGER;
    v_status3 INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT_ENABLE();
    PERFORM DBMS_OUTPUT_PUT_LINE('First line');
    PERFORM DBMS_OUTPUT_PUT_LINE('Second line');

    SELECT line, status INTO v_line1, v_status1 FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);
    RAISE NOTICE '1: line="%", status=%', v_line1, v_status1;

    SELECT line, status INTO v_line2, v_status2 FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);
    RAISE NOTICE '2: line="%", status=%', v_line2, v_status2;

    SELECT line, status INTO v_line3, v_status3 FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);
    RAISE NOTICE '3: line="%", status=%', v_line3, v_status3;

    IF v_line1 = 'First line' AND v_line2 = 'Second line' AND v_line3 IS NULL THEN
        RAISE NOTICE ' GET_LINE работает корректно';
    ELSE
        RAISE EXCEPTION ' Ошибка';
    END IF;

    RAISE NOTICE 'DBMS_OUTPUT_GET_LINE: УСПЕШНО';
END;
$$;
```

DBMS_OUTPUT_GET_LINES

Функция получает массив строк из буфера вывода, куда были отправлены сообщения процедурой PUT_LINE. Используется для эффективного массового считывания нескольких строк из буфера одновременно.

Входные параметры:

- lines (PL/SQL TABLE) – коллекция строк (типа CHARARR), в которую сохраняются извлечённые строки из буфера.
- numlines (NUMBER) – число запрашиваемых строк. Значение параметра обновляется функцией и показывает фактическое количество полученных

строк.

Выходные параметры:

- Функция не возвращает значения. *Типичные сценарии использования:*
- Массовое получение отладочной информации из буфера после завершения длительных операций.
- Оптимизированное извлечение большого количества строк для последующей обработки или отображения.

Примеры запросов:

```
DO $$
DECLARE
    v_lines TEXT[];
    v_numlines INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT_ENABLE();
    PERFORM DBMS_OUTPUT_PUT_LINE('A');
    PERFORM DBMS_OUTPUT_PUT_LINE('B');

    SELECT lines, numlines INTO v_lines, v_numlines
    FROM DBMS_OUTPUT_GET_LINES(ARRAY[]::TEXT[], 10);

    RAISE NOTICE 'Прочитано строк: %', v_numlines;

    IF v_numlines > 0 THEN
        RAISE NOTICE 'Первая строка: "%"', v_lines[1];
    END IF;

    RAISE NOTICE 'DBMS_OUTPUT_GET_LINES: УСПЕШНО';
END;
$$;
```

DBMS_OUTPUT.ENABLE

Процедура включает буфер вывода `DBMS_OUTPUT`, в который записываются строки для последующего чтения.

Входящие параметры:

- Процедура может вызываться без параметров.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Подготовка буфера перед записью строк через `DBMS_OUTPUT.PUT` и `DBMS_OUTPUT.PUT_LINE`.
- Использование встроенного механизма накопления строк для отладки и тестирования.
- Проверка корректности записи и чтения данных из буфера `DBMS_OUTPUT`.

Примеры запросов:

```
DO $$
DECLARE
    v_line VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT_ENABLE();
    PERFORM DBMS_OUTPUT_PUT_LINE('test_enable');

    SELECT line, status INTO v_line, v_status FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);

    RAISE NOTICE 'Прочитано: line="%", status=%', v_line, v_status;

    IF v_line = 'test_enable' AND v_status = 0 THEN
        RAISE NOTICE '□ Буфер успешно включен';
    ELSE
        RAISE EXCEPTION '□ Ошибка';
    END IF;

    RAISE NOTICE 'DBMS_OUTPUT_ENABLE: УСПЕШНО';
END;
$$;
```

DBMS_OUTPUT.NEW_LINE

Процедура завершает текущую строку в буфере `DBMS_OUTPUT`, добавляя перевод строки после вызова `DBMS_OUTPUT.PUT`.

Входящие параметры:

- Нет.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Завершение строки после частичной записи через `DBMS_OUTPUT.PUT`.
- Формирование строк в буфере по частям.
- Тестирование корректной сборки и чтения строк из буфера `DBMS_OUTPUT`.

Примеры запросов:

```
DO $$
DECLARE
    v_line VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT_ENABLE();
    PERFORM DBMS_OUTPUT_PUT('Partial line');
    PERFORM DBMS_OUTPUT_NEW_LINE();

    SELECT line, status INTO v_line, v_status FROM DBMS_OUTPUT_GET_LINE(NULL, NULL);

    RAISE NOTICE 'Прочитано: line="%", status=%', v_line, v_status;

    IF v_line = 'Partial line' THEN
        RAISE NOTICE '□ NEW_LINE завершил строку';
    ELSE
        RAISE EXCEPTION '□ Ошибка';
    END IF;

    RAISE NOTICE 'DBMS_OUTPUT_NEW_LINE: УСПЕШНО';
END;
$$;
```

DBMS_OUTPUT_FLUSH_LINE_BUFFER

Данная функция очищает буфер строки.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VOID – функция не возвращает значения

Типичные сценарии использования:

- Принудительная очистка буфера вывода
- Используется в DBMS_OUTPUT.PUT, DBMS_OUTPUT.PUT_LINE, DBMS_OUTPUT.NEW_LINE

Примеры запросов:

```
SELECT DBMS_OUTPUT_FLUSH_LINE_BUFFER();
```

DBMS_OUTPUT_GET_VERSION

Функция возвращает номер версии пакета DBMS_OUTPUT для вывода сообщений.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка совместимости при отладке
- Диагностика версии

Примеры запросов:

```
SELECT dbms_output_get_version();
```

5.2.2. Пакет DBMS_LOB

DBMS_LOB_LOADFROMFILE

Функция загружает данные из внешнего файла (BFILE) во внутренний BLOB.

Входящие параметры: * dest_lob (BLOB) – целевой BLOB, в который производится загрузка * src_file (BFILE) – внешний файл-источник * amount (NUMBER) – количество байт для загрузки из файла * dest_offset (NUMBER, опционально) – позиция в целевом BLOB, по умолчанию 1 * src_offset (NUMBER, опционально) – позиция в файле-источнике, по умолчанию 1

Выходные параметры:

- BLOB – модифицированный целевой BLOB с загруженными данными *Типичные сценарии использования:*
- Импорт файлов с сервера базы данных в табличные BLOB-поля
- Загрузка бинарных данных из файловой системы
- Частичная загрузка больших файлов с заданного смещения

Примеры запросов:

```
SELECT dbms_lob_loadfromfile(
''::blob,
'TEMP_DIR:test_load_01.dat'::bfile,
11::number
);
```

DBMS_LOB_GETCHUNKSIZE

Функция возвращает размер фрагмента (chunk) для указанного LOB.

Входящие параметры: * lob_loc (BLOB/CLOB) – локатор LOB

Выходные параметры: * NUMBER – размер фрагмента в байтах для BLOB или в символах для CLOB

Типичные сценарии использования: * Оптимизация операций чтения и записи LOB * Определение оптимального размера буфера для работы с LOB

Примеры запросов:

```
SELECT dbms_lob_getchunksize(''::blob);
```

DBMS_LOB_FILEEXISTS

Функция проверяет существование внешнего файла на сервере базы данных.

Входящие параметры: * file_loc (BFILE) – локатор внешнего файла

Выходные параметры: * INTEGER – возвращает 1 если файл существует, 0 если файл не существует или передан NULL

Типичные сценарии использования: * Проверка наличия файла перед операцией LOADFROMFILE * Валидация путей к файлам при импорте данных

Примеры запросов:

```
DO $$
DECLARE
v_result INTEGER;
BEGIN
v_result := dbms_lob_fileexists(bfilename('TEMP_DIR', 'test_file.txt'));
END;
```

DBMS_LOB_FILEGETNAME

Функция извлекает имя псевдонима директории и имя файла из локатора BFILE.

Входящие параметры: * file_loc (BFILE) – локатор внешнего файла

Выходные параметры: * RECORD – запись с полями dir_alias (VARCHAR2) и filename (VARCHAR2)

Типичные сценарии использования: * Получение информации о файле из локатора BFILE * Отладка и логирование при работе с внешними файлами

Примеры запросов:

```
DO $$
DECLARE
v_bfile BFILE;
v_result RECORD;
BEGIN
v_bfile := bfilename('TEMP_DIR', 'test_file.txt');
v_result := dbms_lob_filegetname(v_bfile);
END;
```

DBMS_LOB_FILEISOPEN

Функция проверяет, открыт ли внешний файл в текущей сессии.

Входящие параметры: * file_loc (BFILE) – локатор внешнего файла

Выходные параметры: * INTEGER – возвращает 1 если файл открыт, 0 если файл закрыт или локатор недействителен

Типичные сценарии использования: * Проверка перед закрытием файла * Контроль утечек файловых дескрипторов

Примеры запросов:

```
DO $$
DECLARE
v_bfile BFILE;
v_is_open INTEGER;
BEGIN
v_bfile := bfilename('TEMP_DIR', 'test_file.txt');
PERFORM dbms_lob_fileopen(v_bfile, 0);
v_is_open := dbms_lob_fileisopen(v_bfile);
PERFORM dbms_lob_fileclose(v_bfile);
END;
```

DBMS_LOB_READ

Функция читает данные из LOB начиная с указанного смещения.

Входящие параметры: * lob_loc (BLOB/CLOB/BFILE) – локатор LOB или BFILE * amount (NUMBER) – количество байт или символов для чтения * offset (NUMBER) – позиция в LOB с которой начинается чтение (нумерация с 1) * buffer (тип зависит от LOB) – буфер для данных (опционально, NULL для автоматического создания)

Выходные параметры: * RECORD – запись с полями amount (NUMBER) и buffer (RAW для BLOB/BFILE, VARCHAR2 для CLOB)

Типичные сценарии использования: * Чтение фрагментов больших бинарных объектов * Произвольный доступ к данным внутри LOB * Последовательное чтение CLOB с многобайтными символами

Примеры запросов:

```
DO $$
DECLARE
v_blob BLOB;
v_result record;
BEGIN
v_blob := E'\x48656C6C6F20576F726C64'::BLOB;
v_result := dbms_lob_read(v_blob, 5, 1, NULL);
END;
```

DBMS_LOB_COPY

Функция копирует данные из исходного LOB в целевой LOB.

Входящие параметры:

- dest_lob (BLOB/CLOB) – целевой LOB
- src_lob (BLOB/CLOB) – исходный LOB
- amount (NUMBER) – количество байт или символов для копирования
- dest_offset (NUMBER, опционально) – позиция в целевом LOB, по умолчанию 1
- src_offset (NUMBER, опционально) – позиция в исходном LOB, по умолчанию 1

Выходные параметры:

- BLOB/CLOB – модифицированный целевой LOB

Типичные сценарии использования:

- Дублирование LOB-данных внутри таблиц
- Частичное копирование фрагментов больших объектов
- Перемещение данных между LOB-полями

Примеры запросов:

```
DO $$
DECLARE
v_src_lob BLOB;
v_dest_lob BLOB;
BEGIN
v_src_lob := E'\x4142434445'::BLOB;
v_dest_lob := E'\x'::BLOB;
v_dest_lob := dbms_lob_copy(v_dest_lob, v_src_lob, 5, 1, 1);
END;
```

DBMS_LOB_GETLENGTH

Функция возвращает длину LOB.

Входящие параметры:

- lob_loc (BLOB/CLOB/NCLOB) – локатор LOB

Выходные параметры:

- NUMBER – длина в байтах для BLOB или в символах для CLOB. Возвращает NULL если передан NULL.

Типичные сценарии использования:

- Определение размера LOB перед операциями чтения
- Проверка является ли LOB пустым
- Выделение буфера под чтение на основе реального размера

Примеры запросов:

```
DO $$
DECLARE
v_length NUMERIC;
BEGIN
v_length := dbms_lob_getlength(E'\x48656C6C6F'::BLOB);
END;
```

DBMS_LOB_WRITE

Функция записывает данные в LOB начиная с указанного смещения.

Входящие параметры:

- lob_loc (BLOB/CLOB) – целевой LOB
- amount (NUMBER) – количество байт или символов для записи
- offset (NUMBER) – позиция в LOB с которой начинается запись (нумерация с 1)
- buffer (BLOB/VARCHAR2) – буфер с данными для записи

Выходные параметры: * BLOB/CLOB – модифицированный целевой LOB

Типичные сценарии использования: * Запись данных в LOB с произвольной позиции * Частичное обновление содержимого LOB * Вставка данных в середину существующего LOB

Примеры запросов:

```
DO $$
DECLARE
v_result BLOB;
BEGIN
v_result := dbms_lob_write(E'\x'::BLOB, 10, 1, E'\x48656C6C6F576F726C64'::BLOB);
END;
```

DBMS_LOB_APPEND

Функция добавляет данные в конец LOB.

Входящие параметры:

- dest_lob (BLOB/CLOB) – целевой LOB
- src_lob (BLOB/CLOB/VARCHAR2) – данные для добавления

Выходные параметры: * BLOB/CLOB – модифицированный целевой LOB с добавленными данными

Типичные сценарии использования:

- Последовательное накопление данных в LOB
- Конкатенация LOB-объектов
- Формирование больших текстовых или бинарных данных по частям

Примеры запросов:

```
DO $$
DECLARE
v_result CLOB;
BEGIN
v_result := dbms_lob_append(''::CLOB, 'HelloWorld');
END;
```

DBMS_LOB_HAS_OVERFLOW

Вспомогательная функция для: dbms_lob_compare

Описание: проверяет переполнение диапазона при работе с LOB.

Входящие параметры:

- lob_len (NUMBER) – длина LOB
- offset_1 (NUMBER) – первое смещение
- amount (NUMBER) – количество байт или символов

Выходные параметры:

- BOOLEAN – TRUE если есть переполнение, FALSE в противном случае

Типичные сценарии использования:

- Проверка границ при сравнении LOB
- Предотвращение ошибок выхода за пределы диапазона

Примеры запросов:

```
SELECT dbms_lob_has_overflow(10, 20, 30);
```

DBMS_LOB_GET_VERSION

Функция возвращает номер версии пакета DBMS_LOB для работы с большими объектами.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка доступных функций LOB
- Совместимость приложений

Примеры запросов:

```
SELECT dbms_lob_get_version();
```

5.2.3. Пакет DBMS_ASSERT

DBMS_ASSERT.NOOP

Функция возвращает входную строку без каких-либо изменений. Используется как заглушка для валидации SQL-выражений без фактической проверки.

Входящие параметры:

- input_string (VARCHAR2) – исходная строка, которая будет возвращена без изменений.

Выходные параметры:

- Возвращает исходную строку без модификаций.

Типичные сценарии использования:

- Временная заглушка при разработке для обхода проверок.
- Тестирование логики приложения без реальной валидации.
- Отладка цепочки вызовов функций.
- Замена других функций DBMS_ASSERT в тестовых сценариях.

Примеры запросов:

```
select dbms_assert_noop('abc'::clob);
```

DBMS_ASSERT.ENQUOTE_NAME

Функция заключает имя объекта базы данных в кавычки для формирования корректного SQL-идентификатора.

Входящие параметры:

- name (VARCHAR2) – имя объекта базы данных для заключения в кавычки.
- capitalize (BOOLEAN) – флаг, указывающий преобразование имени к верхнему регистру (TRUE - преобразовать, FALSE - оставить как есть).

Выходные параметры:

- Возвращает строку с именем объекта, заключенным в двойные кавычки.

Типичные сценарии использования:

- Динамическое формирование SQL-запросов с именами объектов.
- Обеспечение корректности идентификаторов в DDL-операциях.
- Работа с именами объектов, содержащими специальные символы или в нижнем регистре.
- Подготовка безопасных SQL-выражений для выполнения.

Примеры запросов:

```
select dbms_assert_enquote_name('abc', false);
```

DBMS_ASSERT.ENQUOTE_LITERAL

Функция заключает строковый литерал в одинарные кавычки для безопасного использования в SQL-запросах.

Входящие параметры:

- str (VARCHAR2) – строковый литерал для заключения в кавычки.

Выходные параметры:

- Возвращает строку, заключенную в одинарные кавычки.

Типичные сценарии использования:

- Безопасное формирование SQL-запросов с строковыми литералами.

- Предотвращение SQL-инъекций при динамическом построении запросов.
- Подготовка значений для условий WHERE в SQL-выражениях.
- Обработка пользовательского ввода перед вставкой в запросы.

Примеры запросов:

```
select dbms_assert_enquote_literal('World');
```

DBMS_ASSERT.SIMPLE_SQL_NAME

Функция проверяет, является ли входная строка допустимым простым SQL-идентификатором, и возвращает её без изменений, если проверка пройдена.

Входящие параметры:

- name (VARCHAR2) – строка для проверки на соответствие правилам именования SQL-идентификаторов.

Выходные параметры:

- Возвращает исходную строку, если она является допустимым SQL-идентификатором.

Типичные сценарии использования:

- Валидация имен таблиц, столбцов и других объектов базы данных.
- Проверка пользовательского ввода перед использованием в динамическом SQL.
- Предотвращение SQL-инъекций через имена объектов.
- Обеспечение корректности формируемых SQL-запросов.

Примеры запросов:

```
select dbms_assert_simple_sql_name('abc_efg');
```

DBMS_ASSERT.QUALIFIED_SQL_NAME

Функция проверяет, является ли входная строка допустимым квалифицированным SQL-именем (состоящим из нескольких частей,

разделенных точками), и возвращает её без изменений, если проверка пройдена.

Входящие параметры:

- name (VARCHAR2) – строка для проверки на соответствие правилам именования квалифицированных SQL-идентификаторов.

Выходные параметры:

- Возвращает исходную строку, если она является допустимым квалифицированным SQL-именем.

Типичные сценарии использования:

- Валидация полных путей к объектам базы данных (например, schema.table.column).
- Проверка сложных идентификаторов перед использованием в динамическом SQL.
- Обеспечение безопасности при работе с именами объектов, содержащих специальные символы.
- Подготовка корректных ссылок на объекты в SQL-запросах.

Примеры запросов:

```
select dbms_assert_qualified_sql_name('abc_efg."Tr34$6-".j@sa');
```

DBMS_ASSERT_SIMPLE_SQL_NAME_LOGIC

Вспомогательная функция для: dbms_assert_qualified_sql_name

Описание: проверяет, что строка является корректным простым SQL-именем.

Входящие параметры:

- name (VARCHAR2) – проверяемая строка
- nullok (BOOLEAN) – разрешено ли значение NULL

Выходные параметры:

- VARCHAR2 – исходная строка, если она является корректным SQL-именем

Типичные сценарии использования:

- Валидация идентификаторов перед использованием в динамическом SQL
- Защита от SQL-инъекций

Примеры запросов:

```
SELECT dbms_assert_simple_sql_name_logic('Name', false);
```

DBMS_ASSERT_GET_VERSION

Функция возвращает номер версии пакета DBMS_ASSERT для валидации и санитизации входных данных.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для безопасности приложений
- Аудит защиты от SQL-инъекций

Примеры запросов:

```
SELECT dbms_assert_get_version();
```

5.2.4. Пакет DBMS_RANDOM

DBMS_RANDOM.INITIALIZE

Процедура инициализирует генератор случайных чисел заданным целочисленным начальным значением.

Входящие параметры:

- seed (BINARY_INTEGER) – целочисленное начальное значение для

инициализации генератора.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:_

- Задание фиксированного начального значения для воспроизводимой последовательности случайных чисел (например, в тестах или демонстрациях).
- Синхронизация состояния генератора в разных сессиях (при использовании одинакового seed).

Примеры запросов:

```
-- Запуск 1 с seed = 5
SELECT dbms_random_initialize(5);
SELECT dbms_random_random();  -- число A
SELECT dbms_random_random();  -- число B
SELECT dbms_random_random();  -- число C
```

— Сброс и повтор с тем же seed = 5 SELECT dbms_random_initialize(5); SELECT dbms_random_random(); — число A (то же самое!) SELECT dbms_random_random(); — число B (то же самое!) SELECT dbms_random_random(); — число C (то же самое!)

DBMS_RANDOM.TERMINATE

Процедура явно завершает работу генератора случайных чисел.

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Явное освобождение ресурсов генератора.

Примеры запросов:

```
\serveroutput on
```

DO $$ DECLARE num1 INTEGER; num2 INTEGER; BEGIN — Инициализация с seed=5 PERFORM dbms_random_initialize(5); num1 := dbms_random_random(); RAISE NOTICE 'После initialize(5): %', num1;

```
-- Завершение работы генератора
PERFORM dbms_random_terminate();
```

```
  -- Попытка сгенерировать число после terminate
  num2 := dbms_random_random();  -- Что вернется?
  RAISE NOTICE 'После terminate: %', num2;
END;
$$;
```

DBMS_RANDOM.NORMAL

Функция возвращает случайное число, имеющее нормальное (гауссово) распределение с заданным средним значением (mean) и стандартным отклонением (sigma).

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- NUMBER – случайное число с нормальным распределением.

Типичные сценарии использования:

- Генерация реалистичных тестовых данных, имитирующих естественные процессы (рост, вес, ошибки измерений).
- Моделирование финансовых показателей, волатильности рынка.
- Создание данных для тестирования статистических алгоритмов и отчетов.
- Симуляция шума или случайных отклонений в научных расчетах.

Примеры запросов:

```
select dbms_random_normal();
```

DBMS_RANDOM.RANDOM

Функция возвращает случайное целое число в диапазоне от -2^31 до 2^31. Устаревшая функция, рекомендуется использовать DBMS_RANDOM.VALUE или DBMS_RANDOM.STRING.

Входящие параметры:

Параметры отсутствуют.

Выходные параметры:

- BINARY_INTEGER – случайное целое число.

Типичные сценарии использования:

- Получение случайного целого числа в устаревшем коде.

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
  num INTEGER;
BEGIN
  PERFORM dbms_random_initialize(5);
  num := dbms_random_random();
  PERFORM dbms_output_put_line('NUM: ' || num);
  PERFORM dbms_random_terminate();
END;
$$;
```

DBMS_RANDOM.SEED

Процедура задает начальное значение (seed) для генератора случайных чисел. Используется для создания воспроизводимой последовательности "случайных" чисел. Может принимать как числовое, так и строковое значение.

Входящие параметры:

- seed (PLS_INTEGER | VARCHAR2) – начальное значение. Может быть целым числом или строкой.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Воспроизведение одной и той же последовательности случайных чисел для целей отладки и тестирования.
- Гарантия идентичной генерации данных в разных сессиях при одном начальном значении.
- Инициализация генератора более случайным значением (например, с помощью строки или текущего времени в микросекундах).

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
  num INTEGER;
BEGIN
  PERFORM dbms_random_seed('3');
  num := dbms_random_random();
  PERFORM dbms_output_put_line('NUM: ' || num);
END;
$$;
```

DBMS_RANDOM.STRING

Функция генерирует случайную строку заданного типа и длины.

Входящие параметры:

- opt (CHAR) – тип генерируемых символов в строке.
- len (NUMBER) – длина генерируемой строки.

Выходные параметры:

- VARCHAR2 – случайная строка заданной длины и типа.

Типичные сценарии использования:

- Генерация временных паролей, токенов доступа, одноразовых кодов.
- Создание уникальных идентификаторов сессий или транзакций.
- Заполнение тестовых данных для строковых полей таблиц.
- Генерация случайных имен файлов, каталогов.
- Создание тестовых данных для проверки обработки строк разного формата.

Примеры запросов:

```
SELECT dbms_random_string('A', 30);

SELECT dbms_random_string('U', 30);

SELECT dbms_random_string('L', 30);

SELECT dbms_random_string('X', 30);

SELECT dbms_random_string('P', 30);
```

DBMS_RANDOM.VALUE

Функция возвращает случайное число типа NUMBER. Может быть вызвана без параметров или с указанием диапазона.

Входящие параметры:

- low (NUMBER) – нижняя граница диапазона (включительно). По умолчанию 0.
- high (NUMBER) – верхняя граница диапазона (исключительно). По умолчанию 1.

Выходные параметры:

- NUMBER – случайное число в диапазоне.

Типичные сценарии использования:

- Генерация случайных дробных чисел для симуляций, игр, тестовых данных.
- Создание случайных процентов, долей, вероятностей.

- Получение случайного порядка записей в выборке (например, ORDER BY DBMS_RANDOM.VALUE).
- Моделирование случайных временных интервалов или задержек.
- Формирование случайных значений для заполнения полей в демонстрационных целях.

Примеры запросов:

```
\serveroutput on

SELECT dbms_random_value();

DO $$
DECLARE
  num INTEGER;
BEGIN
  num := dbms_random_value(10, 20);
  PERFORM dbms_output_put_line('NUM: ' || num);
END;
$$;
```

DBMS_RANDOM_SECONDS_SINCE_EPOCH

Вспомогательная функция для: dbms_random_terminate

Описание: возвращает количество секунд с начала Unix-эпохи.

Входящие параметры:

- Нет параметров

Выходные параметры:

- FLOAT8 – количество секунд с 1970-01-01

Типичные сценарии использования:

- Генерация временных меток для инициализации генератора
- Создание уникальных значений

Примеры запросов:

```
SELECT dbms_random_seconds_since_epoch();
```

DBMS_RANDOM_GET_VERSION

Функция возвращает номер версии пакета DBMS_RANDOM для генерации случайных чисел.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии генератора случайных чисел
- Совместимость алгоритмов

Примеры запросов:

```
SELECT dbms_random_get_version();
```

5.2.5. Пакет DBMS_AQADM

DBMS_AQADM.CREATE_QUEUE_TABLE

Процедура создаёт таблицу очереди для дальнейшего использования механизмом `Advanced Queuing`.

Входящие параметры:

- queue_table (VARCHAR2) – имя создаваемой таблицы очереди.
- queue_payload_type (VARCHAR2) – тип полезной нагрузки сообщений.
- multiple_consumers и другие параметры – дополнительные настройки таблицы очереди.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Подготовка инфраструктуры очередей.
- Создание таблицы для хранения сообщений.
- Инициализация окружения перед созданием очереди.

Примеры запросов:

```
DO $$
DECLARE
    v_qt_name TEXT := 'SYS.NOTIF_QT4';
    v_failed BOOLEAN := FALSE;
BEGIN
    PERFORM DBMS_AQADM_CREATE_QUEUE_TABLE(
        v_qt_name,
        'XMLTYPE',
        NULL,
        NULL,
        false,
        0,
        'test queue table',
        true,
        0,
        0,
        NULL,
        false,
        0
    );

    BEGIN
        PERFORM DBMS_AQADM_CREATE_QUEUE_TABLE(
            v_qt_name,
            'XMLTYPE',
            NULL,
            NULL,
            false,
            0,
            'test queue table',
            true,
            0,
            0,
            NULL,
            false,
            0
        );
```

```
    EXCEPTION
        WHEN OTHERS THEN
            v_failed := TRUE;
    END;

    IF v_failed THEN
        RAISE NOTICE '□ DBMS_AQADM_CREATE_QUEUE_TABLE: Очередь создана, повторное
создание - провал';
    ELSE
        RAISE EXCEPTION '□ DBMS_AQADM_CREATE_QUEUE_TABLE: повторное создание прошло
успешно';
    END IF;
END;
$$;
```

DBMS_AQADM.CREATE_QUEUE

Процедура создаёт очередь на основе ранее созданной таблицы очереди.

Входящие параметры:

- queue_name (VARCHAR2) – имя создаваемой очереди.
- queue_table (VARCHAR2) – имя таблицы очереди.
- queue_type и другие параметры – дополнительные настройки очереди.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Создание логической очереди поверх таблицы очереди.
- Подготовка окружения для постановки и извлечения сообщений.
- Инициализация механизма асинхронного обмена сообщениями.

Примеры запросов:

```
DO $$
DECLARE
    v_qt_name TEXT;
    v_q_name  TEXT;
BEGIN
    v_qt_name := 'SYS.NOTIF_QT_' || substr(md5(clock_timestamp()::text), 1, 8);
```

```
    v_q_name  := 'SYS.NOTIF_Q_'  || substr(md5((clock_timestamp() + interval '1
microsecond')::text), 1, 8);

    PERFORM DBMS_AQADM_CREATE_QUEUE_TABLE(
        v_qt_name,
        'XMLTYPE',
        NULL,
        NULL,
        false,
        0,
        'test queue table',
        true,
        0,
        0,
        NULL,
        false,
        0
    );

    PERFORM DBMS_AQADM_CREATE_QUEUE(
        v_q_name,
        v_qt_name,
        0,
        NULL,
        NULL,
        NULL,
        false,
        'test queue',
        false
    );

    BEGIN
        PERFORM DBMS_AQADM_CREATE_QUEUE(
            v_q_name,
            v_qt_name,
            0,
            NULL,
            NULL,
            NULL,
            false,
            'test queue',
            false
        );

        RAISE EXCEPTION '□ DBMS_AQADM_CREATE_QUEUE: очередь не была создана';
    EXCEPTION
        WHEN OTHERS THEN
```

```
            RAISE NOTICE '□ DBMS_AQADM_CREATE_QUEUE: очередь создана';
    END;
END;
$$;
```

DBMS_AQADM_CREATE_QUEUE2

Создаёт очередь DBMS_AQ.

Входящие параметры:

- queue_name (VARCHAR2) – имя создаваемой очереди
- queue_table (VARCHAR2) – имя таблицы очереди
- magic_value (BINARY_INTEGER) – магическое значение (должно быть фиксированным: 1734829927)
- queue_type (BINARY_INTEGER, опционально) – тип очереди, по умолчанию NORMAL_QUEUE
- max_retries (NUMBER, опционально) – максимальное количество повторных попыток, по умолчанию NULL
- retry_delay (NUMBER, опционально) – задержка между повторными попытками, по умолчанию 0
- retention_time (NUMBER, опционально) – время хранения сообщений, по умолчанию 0
- dependency_tracking (BOOLEAN, опционально) – отслеживание зависимостей, по умолчанию FALSE
- comment (VARCHAR2, опционально) – комментарий к очереди, по умолчанию NULL
- auto_commit (BOOLEAN, опционально) – автоматический коммит, по умолчанию TRUE

Выходные параметры:

- VOID – функция не возвращает значения

Типичные сценарии использования:

- Создание очередей для асинхронной обработки сообщений
- Настройка очередей с параметрами повторных попыток и хранения

Примеры запросов:

```
SELECT DBMS_AQADM_CREATE_QUEUE2('test_q1', 'test_mqt', 1734829927, NULL, NULL, NULL,
NULL, NULL, NULL, NULL);
```

DBMS_AQADM_GET_VERSION

Функция возвращает номер версии пакета DBMS_AQADM для администрирования очередей сообщений.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии перед выполнением административных операций
- Совместимость скриптов администрирования

Примеры запросов:

```
SELECT dbms_aqadm_get_version();
```

===DBMS_AQ

DBMS_AQ.ENQUEUE

Функция помещает сообщение в очередь и возвращает идентификатор сообщения.

Входящие параметры:

- queue_name (VARCHAR2) – имя очереди.
- enqueue_options – параметры постановки сообщения в очередь.
- message_properties – свойства сообщения.
- payload – полезная нагрузка сообщения.

- msgid – параметр идентификатора сообщения.

Выходные параметры:

- Возвращает идентификатор поставленного в очередь сообщения.

Типичные сценарии использования:

- Постановка сообщений в очередь.
- Асинхронная передача данных между компонентами системы.
- Тестирование механизма очередей.

Примеры запросов:

```
DO $$
DECLARE
    v_qt_name TEXT;
    v_q_name  TEXT;
    v_msgid   BYTEA;
BEGIN
    v_qt_name := 'SYS.NOTIF_QT_' || substr(md5(clock_timestamp()::text), 1, 8);
    v_q_name  := 'SYS.NOTIF_Q_'  || substr(md5((clock_timestamp() + interval '1
microsecond')::text), 1, 8);

    PERFORM DBMS_AQADM_CREATE_QUEUE_TABLE(
        v_qt_name,
        'XMLTYPE',
        NULL,
        NULL,
        false,
        0,
        'test queue table',
        true,
        0,
        0,
        NULL,
        false,
        0
    );

    PERFORM DBMS_AQADM_CREATE_QUEUE(
        v_q_name,
        v_qt_name,
        0,
        NULL,
```

```
        NULL,
        NULL,
        false,
        'test queue',
        false
    );

    SELECT t.msgid
      INTO v_msgid
      FROM DBMS_AQ_ENQUEUE(
          v_q_name,
          NULL,
          NULL,
          '<x>121</x>',
          NULL
      ) AS t(msgid BYTEA);

    IF v_msgid IS NOT NULL THEN
        RAISE NOTICE '□ DBMS_AQ_ENQUEUE: сообщение поставлено в очередь';
    ELSE
        RAISE EXCEPTION '□ DBMS_AQ_ENQUEUE: msgid не получен';
    END IF;
END;
$$;
```

DBMS_AQ.DEQUEUE

Функция извлекает сообщение из очереди и возвращает полезную нагрузку и идентификатор сообщения.

Входящие параметры:

- queue_name (VARCHAR2) – имя очереди.
- dequeue_options – параметры извлечения сообщения.
- message_properties – свойства сообщения.
- payload – полезная нагрузка сообщения.
- msgid – идентификатор сообщения.

Выходные параметры:

- Возвращает полезную нагрузку и идентификатор извлечённого сообщения.

Типичные сценарии использования:

- Извлечение сообщений из очереди.
- Обработка асинхронно доставленных данных.
- Проверка корректности работы очередей.

Примеры запросов:

```
DO $$
DECLARE
    v_qt_name  TEXT;
    v_q_name   TEXT;
    v_payload  TEXT;
    v_msgid    BYTEA;
    v_enq_msgid BYTEA;
BEGIN
    v_qt_name := 'SYS.NOTIF_QT_' || substr(md5(clock_timestamp()::text), 1, 8);
    v_q_name  := 'SYS.NOTIF_Q_'  || substr(md5((clock_timestamp() + interval '1
microsecond')::text), 1, 8);

    PERFORM DBMS_AQADM_CREATE_QUEUE_TABLE(
        v_qt_name,
        'XMLTYPE',
        NULL,
        NULL,
        false,
        0,
        'test queue table',
        true,
        0,
        0,
        NULL,
        false,
        0
    );

    PERFORM DBMS_AQADM_CREATE_QUEUE(
        v_q_name,
        v_qt_name,
        0,
        NULL,
        NULL,
        NULL,
        false,
        'test queue',
        false
    );
```

```
    SELECT t.msgid
      INTO v_enq_msgid
      FROM DBMS_AQ_ENQUEUE(
          v_q_name,
          NULL,
          NULL,
          '<x>121</x>',
          NULL
      ) AS t(msgid BYTEA);

    SELECT payload, msgid
      INTO v_payload, v_msgid
      FROM DBMS_AQ_DEQUEUE(
          v_q_name,
          NULL,
          NULL,
          NULL,
          NULL
      );

    IF v_payload = '<x>121</x>' AND v_msgid IS NOT NULL THEN
        RAISE NOTICE '✅ DBMS_AQ_DEQUEUE: сообщение успешно извлечено';
    ELSE
        RAISE EXCEPTION '❌ DBMS_AQ_DEQUEUE: получены некорректные данные payload=%,
msgid=%', v_payload, v_msgid;
    END IF;
END;
$$;
```

DBMS_AQ_FNV1A32

Функция вычисляет хэш FNV1A32 данной строки.

Входящие параметры:

- str (TEXT) – входная строка для вычисления хэша
- len (INT4) – длина строки в байтах

Выходные параметры:

- INT4 – вычисленное хэш-значение FNV1A32

Типичные сценарии использования:

- Вычисление хэша для сообщений при постановке в очередь (DBMS_AQ.ENQUEUE)
- Быстрое хэширование строк для распределения нагрузки

Примеры запросов:

```
SELECT DBMS_AQ_FNV1A32('1234', 4);
```

DBMS_AQ_GET_VERSION

Функция возвращает номер версии пакета DBMS_AQ для работы с очередями сообщений.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии при работе с очередями
- Обеспечение совместимости кода

Примеры запросов:

```
SELECT dbms_aq_get_version();
```

5.2.6. Пакет DBMS_ALERT

DBMS_ALERT.REGISTER

Процедура регистрирует имя алерта в текущей сессии для последующего ожидания сигналов.

Входящие параметры:

- name (VARCHAR2) – имя регистрируемого алерта.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Подготовка сессии к ожиданию событий.
- Подписка на получение конкретного алерта.
- Настройка межсессионного взаимодействия через `DBMS_ALERT`.

Примеры запросов:

```
DO $$
DECLARE
    v_msg VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_ALERT_REGISTER('TEST_REGISTER');

    SELECT message, status
      INTO v_msg, v_status
      FROM DBMS_ALERT_WAITONE('TEST_REGISTER', NULL, NULL, 1);

    IF v_status = 1 THEN
        RAISE NOTICE '□ DBMS_ALERT_REGISTER: алерт зарегистрирован, ожидание
работает';
    ELSE
        RAISE NOTICE '□ DBMS_ALERT_REGISTER: неожиданный статус=%', v_status;
    END IF;

    PERFORM DBMS_ALERT_REMOVE('TEST_REGISTER');
END;
$$;
```

DBMS_ALERT.REMOVE

Процедура удаляет ранее зарегистрированный алерт из текущей сессии.

Входящие параметры:

- name (VARCHAR2) – имя удаляемого алерта.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Отмена регистрации алерта.
- Очистка состояния после завершения теста.
- Управление списком отслеживаемых событий.

Примеры запросов:

```
DO $$
DECLARE
    v_msg VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_ALERT_REGISTER('TEST_REMOVE');
    PERFORM DBMS_ALERT_REMOVE('TEST_REMOVE');

    SELECT message, status
      INTO v_msg, v_status
      FROM DBMS_ALERT_WAITONE('TEST_REMOVE', NULL, NULL, 1);

    IF v_status <> 0 THEN
        RAISE NOTICE '□ DBMS_ALERT_REMOVE: алерт удален';
    ELSE
        RAISE NOTICE '□ DBMS_ALERT_REMOVE: алерт все еще активен';
    END IF;
END;
$$;
```

DBMS_ALERT.SIGNAL

Процедура отправляет сигнал по указанному алерту с текстовым сообщением.

Входящие параметры:

- name (VARCHAR2) – имя алерта.
- message (VARCHAR2) – текст сообщения, передаваемого вместе с сигналом.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Передача уведомлений между сессиями.
- Сигнализация о наступлении событий.
- Тестирование механизма алертов.

Примеры запросов:

```
DO $$
DECLARE
    v_msg VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_ALERT_REGISTER('TEST_SIGNAL');
    PERFORM DBMS_ALERT_SIGNAL('TEST_SIGNAL', 'Тест сообщение');
    COMMIT;

    SELECT message, status
      INTO v_msg, v_status
      FROM DBMS_ALERT_WAITONE('TEST_SIGNAL', NULL, NULL, 2);

    IF v_status = 0 AND v_msg = 'Тест сообщение' THEN
        RAISE NOTICE '□ DBMS_ALERT_SIGNAL: сигнал отправлен';
    ELSE
        RAISE NOTICE '□ DBMS_ALERT_SIGNAL: статус=%, сообщение=%', v_status, v_msg;
    END IF;

    PERFORM DBMS_ALERT_REMOVE('TEST_SIGNAL');
END;
$$;
```

DBMS_ALERT.WAITONE

Функция ожидает сигнал по указанному алерту и возвращает сообщение и статус.

Входящие параметры:

- name (VARCHAR2) – имя ожидаемого алерта.
- message (VARCHAR2) – параметр для получения сообщения.
- status (INTEGER) – параметр для получения статуса.
- timeout (INTEGER) – время ожидания в секундах.

Выходные параметры:

- Возвращает сообщение и статус ожидания.

Типичные сценарии использования:

- Ожидание одного конкретного события.
- Получение сообщения от другого процесса или сессии.
- Реализация простого механизма синхронизации.

Примеры запросов:

```
DO $$
DECLARE
    v_msg VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_ALERT_REGISTER('TEST_WAITONE');
    PERFORM DBMS_ALERT_SIGNAL('TEST_WAITONE', 'WAITONE_OK');
    COMMIT;

    SELECT message, status
      INTO v_msg, v_status
      FROM DBMS_ALERT_WAITONE('TEST_WAITONE', NULL, NULL, 2);

    IF v_status = 0 AND v_msg = 'WAITONE_OK' THEN
        RAISE NOTICE '□ DBMS_ALERT_WAITONE: сообщение получено';
    ELSE
        RAISE NOTICE '□ DBMS_ALERT_WAITONE: статус=%, сообщение=%', v_status, v_msg;
    END IF;

    PERFORM DBMS_ALERT_REMOVE('TEST_WAITONE');
END;
$$;
```

DBMS_ALERT.WAITANY

Функция ожидает сигнал по любому из зарегистрированных алертов и возвращает имя, сообщение и статус.

Входящие параметры:

- name (VARCHAR2) – параметр для получения имени алерта.

- message (VARCHAR2) – параметр для получения сообщения.
- status (INTEGER) – параметр для получения статуса.
- timeout (INTEGER) – время ожидания в секундах.

Выходные параметры:

- Возвращает имя алерта, сообщение и статус ожидания.

Типичные сценарии использования:

- Ожидание нескольких возможных событий.
- Реализация диспетчеризации по имени алерта.
- Синхронизация по первому пришедшему уведомлению.

Примеры запросов:

```
DO $$
DECLARE
    v_name VARCHAR2;
    v_msg VARCHAR2;
    v_status INTEGER;
BEGIN
    PERFORM DBMS_ALERT_REGISTER('TEST_WAITANY_A');
    PERFORM DBMS_ALERT_REGISTER('TEST_WAITANY_B');

    PERFORM DBMS_ALERT_SIGNAL('TEST_WAITANY_B', 'MSG_B');
    COMMIT;

    SELECT name, message, status
      INTO v_name, v_msg, v_status
      FROM DBMS_ALERT_WAITANY(NULL, NULL, NULL, 2);

    IF v_status = 0 AND v_name = 'TEST_WAITANY_B' AND v_msg = 'MSG_B' THEN
        RAISE NOTICE '□ DBMS_ALERT_WAITANY: получен ожидаемый alert';
    ELSE
        RAISE NOTICE '□ DBMS_ALERT_WAITANY: name=%, msg=%, status=%', v_name, v_msg,
v_status;
    END IF;

    PERFORM DBMS_ALERT_REMOVE('TEST_WAITANY_A');
    PERFORM DBMS_ALERT_REMOVE('TEST_WAITANY_B');
END;
$$;
```

DBMS_ALERT.SET_DEFAULTS

Процедура задаёт параметры по умолчанию для механизма алертов.

Входящие параметры:

- sensitivity (INTEGER) – значение параметра по умолчанию для обработки алертов.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Настройка поведения механизма алертов на уровне сессии.
- Подготовка окружения перед тестами `DBMS_ALERT`.
- Централизованная установка параметров ожидания и обработки уведомлений.

Примеры запросов:

```
DO $$
BEGIN
    PERFORM DBMS_ALERT_SET_DEFAULTS(3);
    RAISE NOTICE '□ DBMS_ALERT_SET_DEFAULTS: выполнен без ошибки';
END;
$$;
```

DBMS_ALERT_SET_DEFAULTS_SENS

Вспомогательная функция для: dbms_alert_set_defaults

Описание: устанавливает интервал опроса для механизма ожидания событий.

Входящие параметры:

- value (FLOAT8) – значение интервала опроса в секундах

Выходные параметры:

- VOID – функция не возвращает значения

Типичные сценарии использования:

- Настройка чувствительности механизма оповещений
- Оптимизация частоты проверки событий

Примеры запросов:

```
SELECT dbms_alert_set_defaults_sens(1);
```

DBMS_ALERT_GET_VERSION

Функция возвращает номер версии пакета DBMS_ALERT.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка совместимости версий при использовании сигналов
- Отладка и диагностика

Примеры запросов:

```
SELECT dbms_alert_get_version();
```

5.2.7. Пакет DBMS_DESCRIBE

DBMS_DESCRIBE.DATA_TYPE_TO_ID

Функция возвращает числовой идентификатор типа данных по его имени.

Входящие параметры:

- type_name (VARCHAR2) – имя типа данных.

Выходные параметры:

- Возвращает числовой идентификатор типа данных.

Типичные сценарии использования:

- Получение внутреннего кода типа данных по его имени.
- Использование в механизмах динамического анализа типов.
- Проверка соответствия строкового имени типа его внутреннему представлению.

Примеры запросов:

```
-- 1. dbms_describe_data_type_to_id
-- Назначение: получить числовой идентификатор типа данных по его имени.
-- Результат: код типа VARCHAR2.
SELECT dbms_describe_data_type_to_id('VARCHAR2') AS type_id;
```

DBMS_DESCRIBE.DESCRIBE_PROCEDURE

Функция возвращает описание параметров процедуры или функции.

Входящие параметры:

- object_name (VARCHAR2) – имя процедуры или функции.
- schema_name (VARCHAR2) – имя схемы.
- package_name (VARCHAR2) – имя пакета.
- resolve_synonyms (BOOLEAN) – признак разрешения синонимов.

Выходные параметры:

- Возвращает сведения о параметрах: позиции, имена аргументов, коды типов, значения по умолчанию и режимы параметров.

Типичные сценарии использования:

- Интроспекция сигнатуры процедуры или функции.
- Анализ входных и выходных параметров объектов базы данных.
- Использование в средствах генерации документации и динамического SQL.

Примеры запросов:

```
-- 2. dbms_describe_describe_procedure
-- Назначение: получить описание параметров функции.
-- Результат: позиции, имена, коды типов и режимы параметров функции
dbms_describe_data_type_to_id.
SELECT
    unnest(t.position) as position,
    unnest(t.argument_name) as argument_name,
    unnest(t.datatype) as datatype,
    unnest(t.default_value) as default_value,
    unnest(t.in_out) as in_out
FROM dbms_describe_describe_procedure(
    'dbms_describe_data_type_to_id',
    NULL::varchar2,
    NULL::varchar2,
    false
) AS t(
    overload integer[],
    position integer[],
    level integer[],
    argument_name text[],
    datatype integer[],
    default_value integer[],
    in_out integer[],
    length integer[],
    precision integer[],
    scale integer[],
    radix integer[],
    spare integer[]
);
```

DBMS_DESCRIBE_TYPE_ID_ADV

Вспомогательная функция для: dbms_describe_data_type_to_id

Описание: возвращает идентификатор типа данных с учетом режима поиска.

Входящие параметры:

- type_name (VARCHAR2) – имя типа данных
- lookup_mode (INT4) – режим поиска

Выходные параметры:

- INT4 – идентификатор типа данных

Типичные сценарии использования:

- Получение метаданных о типах данных
- Динамическое определение типов

Примеры запросов:

```
SELECT dbms_describe_type_id_adv('VARCHAR2', 0);
```

DBMS_DESCRIBE_GET_VERSION

Функция возвращает номер версии пакета DBMS_DESCRIBE для получения информации о хранимых процедурах.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии при динамическом описании объектов
- Совместимость метаданных

Примеры запросов:

```
SELECT dbms_describe_get_version();
```

5.2.8. Пакет DBMS_REPORT

DBMS_REPORT.JSON_TO_TEXT

Функция преобразует JSON-документ и JSON-метаданные шаблона в текстовый отчет.

Входящие параметры:

- json_report (CLOB) – исходный JSON-документ с данными отчета.

- json_meta (CLOB) – JSON-документ с описанием структуры итогового текстового представления.

Выходные параметры:

- Возвращает CLOB со сформированным текстовым отчетом.

Типичные сценарии использования:

- Генерация текстовых отчетов по JSON-данным.
- Формирование человекочитаемого представления для логов и выгрузок.
- Преобразование структурированного JSON в шаблонный текст.

Примеры запросов:

```
select dbms_report_json_to_text(
    '{"name": "Иван", "age": 30}'::clob,
    '{
      "nest": [
        {"type": "p", "args": ["Имя: ", "$.name", ", возраст: ", "$.age"]}
      ]
    }'::clob
);
```

```
select dbms_report_json_to_text(
    '{}'::clob,
    '{
      "nest": [
        {"type": "h1", "args": ["Заголовок"], "line": "="}
      ]
    }'::clob
);
```

DBMS_REPORT.ZLIB2BASE64_CLOB

Функция сжимает входную строку алгоритмом zlib и возвращает результат в виде строки Base64 в формате CLOB.

Входящие параметры:

- input_text (VARCHAR2) – исходная строка для сжатия и кодирования.

Выходные параметры:

- Возвращает CLOB со строкой Base64.

Типичные сценарии использования:

- Компактное хранение текстовых данных.
- Подготовка содержимого для передачи по текстовым каналам.
- Формирование переносимых текстовых представлений бинарных данных.

Примеры запросов:

```
select dbms_report_zlib2base64_clob('Hello, World!');
select dbms_report_zlib2base64_clob('Привет, мир!');
```

DBMS_REPORT_EXTRACT_STRING_VALUE

Извлекает строковое значение из строки, содержащей документ JSON.

Входящие параметры:

- _JSON (TEXT) – JSON-документ в виде строки
- KEY (TEXT) – ключ, значение которого необходимо извлечь

Выходные параметры:

- TEXT – строковое значение по указанному ключу

Типичные сценарии использования:

- Парсинг JSON-документов в отчетах
- Используется в DBMS_REPORT.JSON_TO_HTML

Примеры запросов:

```
SELECT DBMS_REPORT_EXTRACT_STRING_VALUE('{"aaa": "bbb"}', 'aaa');
```

DBMS_REPORT_GET_VERSION

Функция возвращает номер версии пакета DBMS_REPORT для генерации отчетов.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для формирования отчетов
- Совместимость форматов вывода

Примеры запросов:

```
SELECT dbms_report_get_version();
```

5.2.9. Пакет DBMS_UTILITY

DBMS_UTILITY.GET_TIME

Функция возвращает текущее системное время в условных тиках, пригодных для измерения интервалов.

Входящие параметры:

- Отсутствуют.

Выходные параметры:

- Возвращает числовое значение времени.

Типичные сценарии использования:

- Замеры длительности операций.
- Профилирование отдельных этапов обработки.

Примеры запросов:

```
select dbms_utility_get_time();
```

DBMS_UTILITY.GET_CPU_TIME

Функция возвращает накопленное процессорное время текущей сессии.

Входящие параметры:

- Отсутствуют.

Выходные параметры:

- Возвращает числовое значение CPU-времени.

Типичные сценарии использования:

- Сравнение стоимости вычислений.
- Профилирование CPU-нагрузки.

Примеры запросов:

```
select dbms_utility_get_cpu_time();
```

DBMS_UTILITY.GET_HASH_VALUE

Функция вычисляет хеш-значение строки в заданном диапазоне.

Входящие параметры:

- name (VARCHAR2) – исходная строка.
- base (NUMBER) – начальное значение диапазона.
- hash_size (NUMBER) – размер диапазона хеширования.

Выходные параметры:

- Возвращает числовое хеш-значение.

Типичные сценарии использования:

- Вычисление служебных хешей.
- Распределение значений по сегментам.

Примеры запросов:

```
select dbms_utility_get_hash_value('test', 0, 1000);
```

DBMS_UTILITY.NAME_TOKENIZE

Функция разбирает составное имя объекта на отдельные компоненты.

Входящие параметры:

- name (VARCHAR2) – имя объекта в составной форме.

Выходные параметры:

- Возвращает набор полей a, b, c, dblink и nextpos.

Типичные сценарии использования:

- Разбор имен вида schema.object@dblink.
- Валидация и анализ имен объектов.

Примеры запросов:

```
DO $$
DECLARE
    v_name VARCHAR2 := 'abc.bb@link';
    v_a VARCHAR2(100) := '';
    v_b VARCHAR2(100) := '';
    v_c VARCHAR2(100) := '';
    v_dblink VARCHAR2(100) := '';
    v_nextpos INTEGER := 0;
BEGIN
    -- Вызываем функцию, передавая все параметры (входные и выходные)
    SELECT * INTO v_a, v_b, v_c, v_dblink, v_nextpos
    FROM DBMS_UTILITY_NAME_TOKENIZE(v_name, v_a, v_b, v_c, v_dblink, v_nextpos);

    RAISE NOTICE 'Исходная строка: %', v_name;
    RAISE NOTICE 'a (первая часть): %', v_a;
    RAISE NOTICE 'b (вторая часть): %', v_b;
    RAISE NOTICE 'c (третья часть): %', v_c;
    RAISE NOTICE 'dblink: %', v_dblink;
    RAISE NOTICE 'nextpos: %', v_nextpos;

    RAISE NOTICE 'DBMS_UTILITY_NAME_TOKENIZE: УСПЕШНО';
END;
$$;
```

DBMS_UTILITY.EXEC_DDL_STATEMENT

Процедура выполняет DDL-оператор, переданный в виде строки.

Входящие параметры:

- statement_text (VARCHAR2) – DDL-команда.

Выходные параметры:

- Не возвращает значения.

Типичные сценарии использования:

- Динамическое создание таблиц и других объектов.
- Выполнение административного DDL из SQL-кода.

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_table_exists BOOLEAN;
    v_table_name VARCHAR := 'employees4';
BEGIN
    -- Выполняем DDL для создания таблицы
    PERFORM DBMS_UTILITY_EXEC_DDL_STATEMENT(
        'CREATE TABLE employees4 (id NUMBER PRIMARY KEY, first_name VARCHAR2(500) NOT
NULL)'
    );
    PERFORM DBMS_OUTPUT_PUT_LINE('Таблица employees4 создана');

    -- Проверяем существование таблицы
    SELECT EXISTS (
        SELECT 1
        FROM information_schema.tables
        WHERE table_name = 'employees4'
    ) INTO v_table_exists;

    IF v_table_exists THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('✓ Таблица employees4 существует');
    ELSE
        PERFORM DBMS_OUTPUT_PUT_LINE('□ Таблица employees4 НЕ существует');
    END IF;

    -- Удаляем таблицу
    PERFORM DBMS_UTILITY_EXEC_DDL_STATEMENT('DROP TABLE employees4');
    PERFORM DBMS_OUTPUT_PUT_LINE('Таблица employees4 удалена');

    -- Проверяем, что таблица удалена
    SELECT EXISTS (
        SELECT 1
        FROM information_schema.tables
        WHERE table_name = 'employees4'
    ) INTO v_table_exists;

    IF NOT v_table_exists THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('✓ Таблица employees4 успешно удалена');
    ELSE
        PERFORM DBMS_OUTPUT_PUT_LINE('□ Таблица employees4 всё ещё существует');
    END IF;
END;
$$;
```

DBMS_UTILITY_MARK_COUNT

Вспомогательная функция для: dbms_utility_name_tokenize

Описание: подсчитывает количество кавычек в строке.

Входящие параметры:

- str (TEXT) – анализируемая строка

Выходные параметры:

- INT4 – количество кавычек в строке

Типичные сценарии использования:

- Разбор идентификаторов с кавычками
- Токенизация SQL-имен

Примеры запросов:

```
SELECT dbms_utility_mark_count('"dl"ed"');
```

DBMS_UTILITY_GET_VERSION

Функция возвращает номер версии пакета DBMS_UTILITY для различных вспомогательных операций.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка совместимости утилит
- Диагностика окружения

Примеры запросов:

```
SELECT dbms_utility_get_version();
```

5.2.10. Пакет DBMS_XMLGEN

DBMS_XMLGEN.NEWCONTEXT

Функция создаёт контекст генерации XML для заданного SQL-запроса.

Входящие параметры:

- query_string (VARCHAR2) – SQL-запрос, результат которого будет преобразован в XML.

Выходные параметры:

- Возвращает идентификатор контекста, используемый в последующих вызовах пакета.

Типичные сценарии использования:

- Инициализация XML-генерации для произвольного SQL-запроса.
- Подготовка контекста перед настройкой параметров формирования XML.
- Использование в сценариях экспорта данных в XML.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    RAISE NOTICE 'Контекст: %', ctx;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.CLOSECONTEXT

Процедура закрывает ранее созданный контекст генерации XML и освобождает связанные с ним ресурсы.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста, созданного функцией `DBMS_XMLGEN.NEWCONTEXT`.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Завершение работы с контекстом генерации XML.
- Освобождение ресурсов после получения XML-документа.
- Корректное завершение многошаговых сценариев работы с `DBMS_XMLGEN`.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);
    RAISE NOTICE 'Контекст закрыт';

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.GETXML

Функция формирует XML-представление результата запроса, связанного с указанным контекстом.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- flags (NUMBER) – дополнительный параметр управления формированием XML.

Выходные параметры:

- Возвращает XML-документ в виде `CLOB`.

Типичные сценарии использования:

- Получение XML из результата SQL-запроса.
- Экспорт строк таблиц в XML.
- Генерация XML после настройки параметров контекста.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETROWSETTAG

Процедура задаёт имя корневого тега для результирующего XML-документа.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.

- rowsettag (VARCHAR2) – имя корневого XML-тега.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Настройка собственного корневого тега XML.
- Подготовка XML к интеграции с внешними системами.
- Формирование XML в соответствии с заданной схемой.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_SETROWSETTAG(ctx, 'root');
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETROWTAG

Процедура задаёт имя XML-тега для отдельной записи результата.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- rowtag (VARCHAR2) – имя тега записи.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Переименование тега строки результата.
- Настройка структуры XML под требования потребителя.
- Формирование XML с пользовательскими именами элементов.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_SETROWTAG(ctx, 'record');
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETNULLHANDLING

Процедура задаёт режим обработки `NULL`-значений при генерации XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- null_handling (NUMBER) – режим обработки `NULL`-значений.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Управление представлением NULL в XML.
- Настройка поведения при экспорте неполных данных.
- Подготовка XML для систем, чувствительных к отсутствующим значениям.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_SETNULLHANDLING(ctx, 2);
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETSKIPROWS

Процедура задаёт количество строк, которые нужно пропустить перед генерацией XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- skip_rows (NUMBER) – количество пропускаемых строк.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Постраничная обработка результатов запроса.

- Пропуск уже обработанных строк.
- Формирование XML по частям.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table ORDER BY id');
    PERFORM DBMS_XMLGEN_SETSKIPROWS(ctx, 2);
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML (пропущено 2): %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETMAXROWS

Процедура ограничивает количество строк, включаемых в результирующий XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- max_rows (NUMBER) – максимальное число строк в результате.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Ограничение размера XML-документа.
- Постраничная выгрузка результатов.

- Тестирование генерации XML на части набора данных.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_SETMAXROWS(ctx, 2);
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML (макс 2): %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETCONVERTSPECIALCHARS

Процедура включает или отключает экранирование специальных символов в XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- convert_special_chars (BOOLEAN) – признак экранирования специальных символов.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Защита XML от некорректных символов.
- Подготовка текстовых данных к безопасному включению в XML.
- Настройка экспорта строк со спецсимволами.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_SETCONVERTSPECIALCHARS(ctx, TRUE);
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETINDENTATIONWIDTH

Процедура задаёт ширину отступов при форматированном выводе XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- indentation_width (NUMBER) – количество пробелов для одного уровня отступа.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Повышение читаемости XML.
- Подготовка XML для визуального анализа.
- Настройка форматированного представления результата.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_SETINDENTATIONWIDTH(ctx, 5);
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETPRETTYPRINTING

Процедура включает или отключает форматированный вывод XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- pretty_printing (BOOLEAN) – признак включения форматированного вывода.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Форматирование XML для чтения человеком.
- Подготовка XML для отладки и анализа.
- Управление компактным или читаемым представлением результата.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    PERFORM DBMS_XMLGEN_SETPRETTYPRINTING(ctx, TRUE);
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.GETNUMROWSPROCESSED

Функция возвращает количество строк, обработанных при последнем формировании XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.

Выходные параметры:

- Возвращает количество обработанных строк.

Типичные сценарии использования:

- Контроль объёма обработанных данных.
- Реализация постраничной обработки.
- Мониторинг работы генератора XML.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
    numproc NUMBER;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    numproc := DBMS_XMLGEN_GETNUMROWSPROCESSED(ctx);
    RAISE NOTICE 'Обработано строк: %', numproc;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETBINDVALUE

Процедура задаёт значение связанной переменной, используемой в SQL-запросе контекста.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- bind_name (VARCHAR2) – имя параметра привязки.
- bind_value (VARCHAR2) – значение параметра привязки.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Подстановка значений в параметризованный SQL-запрос.
- Генерация XML для разных условий выборки без пересоздания запроса.
- Повторное использование контекста с разными входными параметрами.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table WHERE id > :min_id');
    PERFORM DBMS_XMLGEN_SETBINDVALUE(ctx, 'min_id', '1');
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'XML: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.RESTARTQUERY

Процедура перезапускает выполнение запроса, связанного с контекстом.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Повторное выполнение одного и того же запроса.
- Повторная генерация XML без пересоздания контекста.
- Возврат к началу обработки данных.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
    numproc NUMBER;
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table');

    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    numproc := DBMS_XMLGEN_GETNUMROWSPROCESSED(ctx);
    RAISE NOTICE 'Первый раз: % строк', numproc;

    PERFORM DBMS_XMLGEN_RESTARTQUERY(ctx);

    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    numproc := DBMS_XMLGEN_GETNUMROWSPROCESSED(ctx);
    RAISE NOTICE 'После RESTART: % строк', numproc;

    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN.SETXSLT

Процедура задаёт XSLT-преобразование, которое будет применено к сформированному XML.

Входящие параметры:

- ctx (NUMBER) – идентификатор контекста генерации XML.
- xsl_doc (CLOB) – XSLT-документ, применяемый к результирующему XML.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Преобразование XML в текстовый или иной формат.
- Дополнительная обработка результата без изменения исходного SQL.
- Настройка пользовательского представления XML-данных.

Примеры запросов:

```
DO $$
DECLARE
    ctx NUMBER := 0;
    xml_result CLOB;
    v_xsl CLOB := '<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" encoding="UTF-8"/>
    <xsl:template match="/">
        <xsl:apply-templates select="root/record"/>
    </xsl:template>
    <xsl:template match="record">
        <xsl:value-of select="name"/>
        <xsl:text> (</xsl:text>
        <xsl:value-of select="id"/>
        <xsl:text>)</xsl:text>
    </xsl:template>
</xsl:stylesheet>';
BEGIN
    CREATE TEMP TABLE my_table (id INTEGER, name TEXT);
    INSERT INTO my_table VALUES (1, NULL), (2, 'Иван'), (3, 'Мария'), (4, 'Сергей '' "
<> alles');

    ctx := DBMS_XMLGEN_NEWCONTEXT('SELECT * FROM my_table ORDER BY id');
    PERFORM DBMS_XMLGEN_SETROWSETTAG(ctx, 'root');
    PERFORM DBMS_XMLGEN_SETROWTAG(ctx, 'record');
    PERFORM DBMS_XMLGEN_SETXSLT(ctx, v_xsl);
    xml_result := DBMS_XMLGEN_GETXML(ctx, 0);
    RAISE NOTICE 'Результат XSLT: %', xml_result;
    PERFORM DBMS_XMLGEN_CLOSECONTEXT(ctx);

    DROP TABLE my_table;
END;
$$;
```

DBMS_XMLGEN_CLOSECONTEXT_INTERNAL

Вспомогательная функция для: dbms_xmlgen_getxml_ovr3

Описание: закрывает XML-контекст и освобождает ресурсы.

Входящие параметры:

- context_handle (NUMBER) – дескриптор XML-контекста
- cleanup_mode (INT4) – режим очистки

Выходные параметры:

- VOID – функция не возвращает значения

Типичные сценарии использования:

- Освобождение ресурсов после генерации XML
- Завершение работы с XML-контекстом

Примеры запросов:

```
PERFORM dbms_xmlgen_closecontext_internal(1, 1);
```

DBMS_XMLGEN_GET_VERSION

Функция возвращает номер версии пакета DBMS_XMLGEN для генерации XML из SQL-запросов.

Входящие параметры:

- Нет параметров

Выходные параметры

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для XML-функций
- Совместимость форматов

Примеры запросов:

```
SELECT dbms_xmlgen_get_version();
```

5.2.11. Пакет DBMS_CRYPTO

DBMS_CRYPTO.HASH

Функция вычисляет криптографический хеш для значения типа RAW, CLOB или BLOB с использованием заданного алгоритма.

Входящие параметры:

- src (RAW | CLOB | BLOB) – исходное значение.
- typ (NUMBER) – код алгоритма хеширования.

Выходные параметры:

- Возвращает RAW со значением хеша.

Типичные сценарии использования:

- Контроль целостности данных.
- Получение отпечатков содержимого.
- Хеширование текстовых и бинарных значений.

Примеры запросов:

```
DO $$
DECLARE
    str RAW(100) := 'abcd'::RAW;
    res RAW(100);
    hash_type INTEGER;
BEGIN
    RAISE NOTICE 'Исходные данные: %', str;
    RAISE NOTICE '========================================';

    -- HASH_MD4 = 2
    hash_type := 2;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_MD4 (2): %', res;

    -- HASH_MD5 = 3
    hash_type := 3;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_MD5 (3): %', res;

    -- HASH_SH1 = 4
```

```
    hash_type := 4;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_SH1 (4): %', res;

    -- HASH_SHA1 = 5
    hash_type := 5;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_SHA1 (5): %', res;

    -- HASH_MD5 = 6 (другой вариант)
    hash_type := 6;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_MD5_ALT (6): %', res;

    -- HASH_SH256 = 7
    hash_type := 7;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_SH256 (7): %', res;

    -- HASH_SHA256 = 8
    hash_type := 8;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_SHA256 (8): %', res;

    -- HASH_SH384 = 9
    hash_type := 9;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_SH384 (9): %', res;

    -- HASH_SHA384 = 10
    hash_type := 10;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_SHA384 (10): %', res;

    -- HASH_SH512 = 11
    hash_type := 11;
    res := DBMS_CRYPTO_HASH(str, hash_type);
    RAISE NOTICE 'HASH_SH512 (11): %', res;

    RAISE NOTICE '========================================';
    RAISE NOTICE 'DBMS_CRYPTO_HASH: УСПЕШНО';
END;
$$;
```

DBMS_CRYPTO_HASH_CFG

Вспомогательная функция для: dbms_crypto_hash

Описание: вычисляет хеш с учетом профиля совместимости.

Входящие параметры:

- src (RAW | CLOB | BLOB) – исходные данные для хеширования
- typ (NUMBER) – тип алгоритма хеширования
- profile_mask (INT4) – маска профиля совместимости

Выходные параметры:

- RAW – вычисленное хеш-значение

Типичные сценарии использования:

- Вычисление контрольных сумм с учетом обратной совместимости
- Криптографическая защита данных

Примеры запросов:

```
SELECT dbms_crypto_hash_cfg('abcd'::RAW, 2, 2);
```

DBMS_CRYPTO_GET_VERSION

Функция возвращает номер версии пакета DBMS_CRYPTO для криптографических операций.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка доступных криптографических функций
- Совместимость шифрования между средами

Примеры запросов:

```
SELECT dbms_crypto_get_version();
```

5.2.12. Пакет DBMS_HASH

DBMS_HASH.HASH

Функция вычисляет хеш для значения типа RAW, CLOB или BLOB.

Входящие параметры:

- src (RAW | CLOB | BLOB) – исходное значение.
- typ (NUMBER) – код алгоритма хеширования.

Выходные параметры:

- Возвращает RAW со значением хеша.

Типичные сценарии использования:

- Неблокирующее хеширование текстовых и бинарных значений.
- Сравнение данных по хеш-отпечаткам.

Примеры запросов:

```
DO $$
DECLARE
    str RAW(100) := 'abcd'::RAW;
    res RAW(100);
BEGIN
    RAISE NOTICE 'Исходные данные (RAW): %', str;

    res := DBMS_HASH_HASH(str, 1);
    RAISE NOTICE 'Хэш (тип 1): %', res;

    RAISE NOTICE 'DBMS_HASH_HASH (RAW): УСПЕШНО';
END;
$$;
```

```
DO $$
DECLARE
    str TEXT := 'abcd';
    res RAW(100);
BEGIN
    RAISE NOTICE 'Исходные данные (CLOB/TEXT): %', str;

    res := DBMS_HASH_HASH(str::CLOB, 1);
    RAISE NOTICE 'Хэш (тип 1): %', res;

    RAISE NOTICE 'DBMS_HASH_HASH (CLOB): УСПЕШНО';
END;
$$;
```

```
DO $$
DECLARE
    str BLOB;
    res RAW(100);
BEGIN
    -- Преобразуем строку в BLOB
    str := 'abcd'::BYTEA;
    RAISE NOTICE 'Исходные данные (BLOB): %', str;

    res := DBMS_HASH_HASH(str, 1);
    RAISE NOTICE 'Хэш (тип 1): %', res;

    RAISE NOTICE 'DBMS_HASH_HASH (BLOB): УСПЕШНО';
END;
$$;
```

DBMS_HASH_HASH_PROFILE

Вспомогательная функция для: dbms_hash_hash

Описание: вычисляет хеш с учетом профиля подготовки входных данных.

Входящие параметры:

- src (RAW | CLOB | BLOB) – исходные данные
- typ (NUMBER) – тип алгоритма хеширования
- digest_profile (INT4) – профиль подготовки данных

Выходные параметры:

- RAW – вычисленное хеш-значение

Типичные сценарии использования:

- Вычисление хешей с настраиваемой□□□ данных
- Сравнение больших объемов данных

Примеры запросов:

```
SELECT dbms_hash_hash_profile('abcd'::RAW, 1, 1);
```

Функция DBMS_HASH_GET_VERSION

Функция возвращает номер версии пакета DBMS_HASH для хэширования данных.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для хэш-функций
- Совместимость алгоритмов хэширования

Примеры запросов:

```
SELECT dbms_hash_get_version();
```

5.2.13. Пакет DBMS_ODCI

DBMS_ODCI.ESTIMATE_CPU_UNITS

Функция оценивает число CPU-единиц для указанной продолжительности работы.

Входящие параметры:

- seconds (NUMBER) – длительность в секундах.

Выходные параметры:

- Возвращает числовую оценку CPU-единиц.

Типичные сценарии использования:

- Оценка ресурсоемкости операций.
- Тестирование линейности расчета CPU-стоимости.

Примеры запросов:

```
DO $$
DECLARE
    v_result NUMBER;
BEGIN
    RAISE NOTICE 'Тест 1. Базовый вызов: 1 секунда';

    v_result := DBMS_ODCI_ESTIMATE_CPU_UNITS(1);
    RAISE NOTICE 'Результат: %', v_result;

    IF v_result IS NOT NULL AND v_result > 0 THEN
        RAISE NOTICE '✓ Функция вернула положительное значение';
    ELSE
        RAISE NOTICE '✗ Результат: % (ожидалось > 0)', v_result;
    END IF;

    RAISE NOTICE 'DBMS_ODCI_ESTIMATE_CPU_UNITS (базовый): УСПЕШНО';
END;
$$;
```

```
DO $$
DECLARE
    v_result NUMBER;
BEGIN
    RAISE NOTICE 'Тест 3. Дробное значение: 0.25';

    v_result := DBMS_ODCI_ESTIMATE_CPU_UNITS(0.25);
    RAISE NOTICE 'Результат: %', v_result;

    IF v_result IS NOT NULL THEN
        RAISE NOTICE '□ Функция обработала дробное значение';
    ELSE
        RAISE EXCEPTION '□ Функция вернула NULL';
    END IF;

    RAISE NOTICE 'DBMS_ODCI_ESTIMATE_CPU_UNITS (дробное): УСПЕШНО';
END;
$$;
```

DBMS_ODCI_GET_SINGLE_CPU_MHZ

Вспомогательная функция для: dbms_odci_estimate_cpu_units

Описание: возвращает частоту одного CPU в MHz.

Входящие параметры:

- Нет параметров

Выходные параметры:

- FLOAT8 – частота процессора в мегагерцах

Типичные сценарии использования:

- Оценка вычислительных ресурсов
- Оптимизация выполнения запросов

Примеры запросов:

```
SELECT dbms_odci_get_single_cpu_mhz();
```

DBMS_ODCI_GET_VERSION

Функция возвращает номер версии пакета DBMS_ODCI для расширения возможностей оптимизатора запросов.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для кардинальных функций
- Отладка расширений оптимизатора

Примеры запросов:

```
SELECT dbms_odci_get_version();
```

5.2.14. Пакет OWA

INIT_CGI_ENV

Функция инициализирует CGI-контекст запроса для последующей работы пакетов `OWA_CACHE`, `OWA_COOKIE` и `OWA_UTIL`.

Входящие параметры:

- `num_cgi_vars (INTEGER)` - количество CGI-параметров.
- `param_name (TEXT[])` - массив имён CGI-параметров.
- `param_val (TEXT[])` - массив значений CGI-параметров.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Подготовка тестового CGI-окружения перед вызовом `OWA_COOKIE.SEND`.
- Эмуляция HTTP-заголовков перед проверкой поведения `OWA_CACHE`.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY['REQUEST_PROTOCOL', 'HTTP_IF_NONE_MATCH'];
    v_values := ARRAY['HTTP/1.1', 'etag-1'];

    PERFORM owa_init_cgi_env(2, v_names, v_values);

    RAISE NOTICE 'OWA.INIT_CGI_ENV: УСПЕШНО';
END;
$$;
```

5.2.15. Пакет OWA_COOKIE

SEND

Функция добавляет cookie в заголовки ответа текущего OWA-контекста.

Входящие параметры:

- `name (VARCHAR2)` - имя cookie.
- `value (VARCHAR2)` - значение cookie.
- `expires (TIMESTAMPTZ)` - дата истечения cookie.
- `path (VARCHAR2)` - путь действия cookie.
- `domain (VARCHAR2)` - домен cookie.
- `secure (VARCHAR2)` - признак secure-cookie.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Установка cookie сессии перед выводом страницы.
- Формирование `Set-Cookie` с дополнительными атрибутами `expires`, `path` и `domain`.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cookie_send(
        name    => 'session_id',
        value   => 'abc123',
        path    => '/app',
        domain  => 'example.com',
        secure  => 'TRUE'
    );
    PERFORM owa_util_showpage();

    RAISE NOTICE 'OWA_COOKIE.SEND: УСПЕШНО';
END;
$$;
```

5.2.16. Пакет OWA_CACHE

SET_CACHE

Функция сохраняет ETag и уровень кеширования в текущем OWA-контексте.

Входящие параметры:

- `p_etag (VARCHAR2)` - значение ETag.
- `p_level (VARCHAR2)` - уровень кеширования, например `USER` или `SYSTEM`.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Подготовка заголовков валидации ответа.
- Настройка кеширования HTML-страницы перед `OWA_UTIL.SHOWPAGE`.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cache_set_cache('etag-101', 'USER');
    PERFORM owa_util_showpage();

    RAISE NOTICE 'OWA_CACHE.SET_CACHE: УСПЕШНО';
END;
$$;
```

SET_EXPIRES

Функция задаёт заголовки кеширования по времени жизни ответа.

Входящие параметры:

- `p_expires (NUMBER)` - срок жизни кеша в минутах.
- `p_level (VARCHAR2)` - уровень кеширования.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Назначение TTL для ответов, которые можно хранить ограниченное время.
- Настройка пользовательского или системного уровня кеширования.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cache_set_expires(60, 'USER');

    RAISE NOTICE 'OWA_CACHE.SET_EXPIRES: УСПЕШНО';
END;
$$;
```

SET_NOT_MODIFIED

Функция переводит ответ в состояние `304 Not Modified`, если входные параметры запроса совпадают с текущими данными кеша.

Входящие параметры:

- У метода нет входящих параметров.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Возврат условного ответа без тела при совпадении ETag.
- Обработка запроса с заголовком `If-None-Match`.

Примеры запросов:

```
DO $$
DECLARE
    v_names  TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY['HTTP_IF_NONE_MATCH', 'REQUEST_PROTOCOL'];
    v_values := ARRAY['etag-101', 'HTTP/1.1'];

    PERFORM owa_init_cgi_env(2, v_names, v_values);
    PERFORM owa_cache_set_cache('etag-101', 'USER');
    PERFORM owa_cache_set_not_modified();

    RAISE NOTICE 'OWA_CACHE.SET_NOT_MODIFIED: УСПЕШНО';
END;
$$;
```

SET_REQUEST_ETAG

Функция принудительно задаёт входной ETag запроса в текущем OWA-контексте.

Входящие параметры:

- `p_etag (VARCHAR2)` - входной ETag запроса.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Подмена `If-None-Match` в тестовом контуре.
- Подготовка сценариев сравнения с `OWA_CACHE.SET_CACHE`.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cache_set_request_etag('etag-202');

    RAISE NOTICE 'ETag=%', owa_cache_get_etag();
    RAISE NOTICE 'OWA_CACHE.SET_REQUEST_ETAG: УСПЕШНО';
END;
$$;
```

SET_REQUEST_LEVEL

Функция принудительно задаёт уровень кеширования входного запроса.

Входящие параметры:

- `p_level (VARCHAR2)` - уровень кеширования запроса.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Эмуляция входного `OWA_CACHE_LEVEL`.
- Подготовка сценариев проверки `OWA_CACHE.GET_LEVEL`.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cache_set_request_level('SYSTEM');

    RAISE NOTICE 'Level=%', owa_cache_get_level();
    RAISE NOTICE 'OWA_CACHE.SET_REQUEST_LEVEL: УСПЕШНО';
END;
$$;
```

GET_ETAG

Функция возвращает текущий входной ETag из OWA-контекста.

Входящие параметры:

- У метода нет входящих параметров.

Выходные параметры:

- Возвращает `VARCHAR2` со значением ETag.

Типичные сценарии использования:

- Проверка того, какой ETag был передан в CGI-контекст.
- Использование в диагностических и тестовых сценариях.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
    v_etag VARCHAR2;
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cache_set_request_etag('etag-303');
    v_etag := owa_cache_get_etag();

    RAISE NOTICE 'Прочитано: etag="%"', v_etag;
    RAISE NOTICE 'OWA_CACHE.GET_ETAG: УСПЕШНО';
END;
$$;
```

GET_LEVEL

Функция возвращает текущий уровень кеширования из OWA-контекста.

Входящие параметры:

- У метода нет входящих параметров.

Выходные параметры:

- Возвращает `VARCHAR2` со значением уровня кеширования.

Типичные сценарии использования:

- Проверка уровня, установленного через CGI-параметры или `SET_REQUEST_LEVEL`.
- Отладка логики ветвления по `USER` или `SYSTEM`.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
    v_level VARCHAR2;
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cache_set_request_level('USER');
    v_level := owa_cache_get_level();

    RAISE NOTICE 'Прочитано: level="%"', v_level;
    RAISE NOTICE 'OWA_CACHE.GET_LEVEL: УСПЕШНО';
END;
$$;
```

SET_SURROGATE_CONTROL

Функция задаёт заголовок `Surrogate-Control` в OWA-контексте.

Входящие параметры:

- `p_value (VARCHAR2)` - значение заголовка `Surrogate-Control`.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Управление кешированием на стороне proxy или CDN.
- Подготовка специализированных HTTP-заголовков для промежуточных узлов.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY['REQUEST_PROTOCOL'];
    v_values := ARRAY['HTTP/1.1'];

    PERFORM owa_init_cgi_env(1, v_names, v_values);
    PERFORM owa_cache_set_surrogate_control('content="ORA:20"');
    PERFORM owa_util_showpage();

    RAISE NOTICE 'OWA_CACHE.SET_SURROGATE_CONTROL: УСПЕШНО';
END;
$$;
```

DISABLE

Функция отключает кеширование для текущего OWA-запроса.

Входящие параметры:

- У метода нет входящих параметров.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Принудительное отключение кеширования перед выводом динамического ответа.
- Сброс ранее установленных параметров кеширования.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM owa_cache_set_cache('etag-disable', 'USER');
    PERFORM owa_cache_disable();
    PERFORM owa_util_showpage();

    RAISE NOTICE 'OWA_CACHE.DISABLE: УСПЕШНО';
END;
$$;
```

5.2.17. Пакет OWA_UTIL

SHOWPAGE

Функция завершает формирование OWA-ответа и выводит накопленные заголовки и тело страницы.

Входящие параметры:

- У метода нет входящих параметров.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Финальный вывод страницы после вызовов `HTP.P`, `OWA_COOKIE.SEND` и `OWA_CACHE`.
- Сброс буфера заголовков и HTML-тела в тестовых сценариях.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
BEGIN
    v_names := ARRAY[]::TEXT[];
    v_values := ARRAY[]::TEXT[];

    PERFORM owa_init_cgi_env(0, v_names, v_values);
    PERFORM htp_p_ovr1('<html><body>Hello</body></html>');
    PERFORM owa_util_showpage();

    RAISE NOTICE 'OWA_UTIL.SHOWPAGE: УСПЕШНО';
END;
$$;
```

GET_CGI_ENV

Функция возвращает значение CGI-параметра из текущего OWA-контекста.

Входящие параметры:

- `param_name (VARCHAR2)` - имя CGI-переменной.

Выходные параметры:

- Возвращает `VARCHAR2` со значением найденного CGI-параметра.

Типичные сценарии использования:

- Чтение `REQUEST_PROTOCOL`, `HTTP_IF_NONE_MATCH` и других входных параметров.
- Отладка наполнения CGI-контекста после `OWA.INIT_CGI_ENV`.

Примеры запросов:

```
DO $$
DECLARE
    v_names TEXT[];
    v_values TEXT[];
    v_protocol VARCHAR2;
BEGIN
    v_names := ARRAY['REQUEST_PROTOCOL'];
    v_values := ARRAY['HTTP/1.1'];

    PERFORM owa_init_cgi_env(1, v_names, v_values);
    v_protocol := owa_util_get_cgi_env('REQUEST_PROTOCOL');

    RAISE NOTICE 'Прочитано: protocol="%"', v_protocol;
    RAISE NOTICE 'OWA_UTIL.GET_CGI_ENV: УСПЕШНО';
END;
$$;
```

5.2.18. Пакет DBMS_JSON

GET_VIEW_SQL

Функция формирует DDL-текст для `VIEW` или `MATERIALIZED VIEW` на основе переданного `data guide`.

Входящие параметры:

- `viewname (VARCHAR2)` - имя представления.
- `tablename (VARCHAR2)` - имя исходной таблицы.
- `jcolname (VARCHAR2)` - имя JSON-колонки.
- `dataguide (JSONB)` - описание полей JSON.
- `materialize (BOOLEAN)` - режим генерации materialized view.
- `mvrefreshmode (INTEGER)` - режим refresh materialized view.
- `path (VARCHAR2)` - корневой JSON path.
- `resolvenameconflicts (BOOLEAN)` - разрешение конфликтов имён.
- `colnameprefix (VARCHAR2)` - префикс имён колонок.
- `mixedcasecolumns (BOOLEAN)` - сохранять ли регистр имён колонок.

Выходные параметры:

- Возвращает `CLOB` с текстом DDL либо `SELECT`.

Типичные сценарии использования:

- Генерация SQL без немедленного создания представления.
- Предпросмотр структуры будущего `VIEW` или `MATERIALIZED VIEW`.

Примеры запросов:

```
DO $$
DECLARE
    v_dg  JSONB;
    v_res CLOB;
BEGIN
    v_dg := '[{"path":"$.id","type":"number","column":"id"}]'::jsonb;

    v_res := dbms_json_get_view_sql('V_JSON_AUTO', 'T_JSON_AUTO', 'J1', v_dg);

    RAISE NOTICE 'DDL: %', v_res;
    RAISE NOTICE 'DBMS_JSON.GET_VIEW_SQL: УСПЕШНО';
END;
$$;
```

CREATE_VIEW

Функция создаёт `VIEW` или `MATERIALIZED VIEW` по переданному `data guide`.

Входящие параметры:

- `viewname (VARCHAR2)` - имя представления.
- `tablename (VARCHAR2)` - имя исходной таблицы.
- `jcolname (VARCHAR2)` - имя JSON-колонки.
- `dataguide (JSONB)` - описание полей JSON.
- `resourcepath (VARCHAR2)` - служебный параметр совместимости.
- `materialize (BOOLEAN)` - создавать materialized view.
- `mvrefreshmode (INTEGER)` - режим refresh.
- `path (VARCHAR2)` - корневой JSON path.
- `resolvenameconflicts (BOOLEAN)` - разрешение конфликтов имён.
- `colnameprefix (VARCHAR2)` - префикс имён колонок.

- `mixedcasecolumns (BOOLEAN)` - сохранять ли регистр имён колонок.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Автоматическое создание проекционного представления по JSON-документам.
- Быстрое построение view для выборок по JSON-полям.

Примеры запросов:

```
DO $$
DECLARE
    v_dg JSONB;
BEGIN
    v_dg := '[{"path":"$.id","type":"number","column":"id"}]'::jsonb;

    EXECUTE 'DROP TABLE IF EXISTS t_json_auto';
    EXECUTE 'CREATE TABLE t_json_auto (j1 jsonb)';
    EXECUTE $$INSERT INTO t_json_auto VALUES ('{"id": 10}'::jsonb)$$;

    PERFORM dbms_json_create_view('v_json_auto', 't_json_auto', 'j1', v_dg);

    RAISE NOTICE 'DBMS_JSON.CREATE_VIEW: УСПЕШНО';
END;
$$;
```

RENAME_COLUMN

Функция сохраняет предпочтительное имя колонки для указанного JSON path.

Входящие параметры:

- `tablename (VARCHAR2)` - имя таблицы.
- `jcolname (VARCHAR2)` - имя JSON-колонки.
- `path (VARCHAR2)` - JSON path до поля.
- `type (INTEGER)` - код JSON-типа.
- `preferred_name (VARCHAR2)` - новое имя колонки.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Подготовка читаемых имён колонок до генерации view.
- Разрешение неоднозначных или неудобных имён JSON-полей.

Примеры запросов:

```
DO $$
BEGIN
    EXECUTE 'DROP TABLE IF EXISTS t_rename';
    EXECUTE 'CREATE TABLE t_rename (jcol jsonb)';

    PERFORM dbms_json_rename_column('t_rename', 'jcol', '$.id', 3, 'renamed_id');

    RAISE NOTICE 'DBMS_JSON.RENAME_COLUMN: УСПЕШНО';
END;
$$;
```

CREATE_VIEW_ON_PATH

Функция создаёт представление только для указанного поддерева JSON path.

Входящие параметры:

- `viewname (VARCHAR2)` - имя представления.
- `tablename (VARCHAR2)` - имя таблицы.
- `jcolname (VARCHAR2)` - имя JSON-колонки.
- `path (VARCHAR2)` - JSON path поддерева.
- `frequency (INTEGER)` - порог частоты поля.

Выходные параметры:

- Функция не возвращает значения.

Типичные сценарии использования:

- Создание узкоспециализированного представления по вложенному JSON-объекту.

- Проекция только нужного фрагмента документа без полного разворачивания.

Примеры запросов:

```
DO $$
BEGIN
    EXECUTE 'DROP TABLE IF EXISTS t_json_path';
    EXECUTE 'CREATE TABLE t_json_path (j1 jsonb)';
    EXECUTE $$INSERT INTO t_json_path VALUES
('{"info":{"label":"ok","code":10}}'::jsonb)$$;

    PERFORM dbms_json_create_view_on_path('v_json_path', 't_json_path', 'j1',
'$.info');

    RAISE NOTICE 'DBMS_JSON.CREATE_VIEW_ON_PATH: УСПЕШНО';
END;
$$;
```

5.2.19. Пакет DBMS_JSON_SCHEMA

IS_VALID

Функция проверяет JSON-данные на соответствие указанной JSON Schema.

Входящие параметры:

- `json_data (JSONB)` - проверяемые JSON-данные.
- `json_schema (JSONB)` - JSON Schema.
- `raise_err (BINARY_INTEGER)` - режим выброса ошибки при невалидном результате.

Выходные параметры:

- Возвращает `BINARY_INTEGER`: `1`, если данные валидны, иначе `0`.

Типичные сценарии использования:

- Валидация входного JSON перед загрузкой в таблицу.
- Проверка JSON-документов в функциях и процедурах.

Примеры запросов:

```
DO $$
DECLARE
    v_res BINARY_INTEGER;
BEGIN
    v_res := dbms_json_schema_is_valid(
        '{"id": 1}'::jsonb,
        '{"type":"object","properties":{"id":{"type":"number"}}}'::jsonb
    );

    RAISE NOTICE 'Прочитано: result=%', v_res;
    RAISE NOTICE 'DBMS_JSON_SCHEMA.IS_VALID: УСПЕШНО';
END;
$$;
```

IS_SCHEMA_VALID

Функция проверяет, является ли переданный JSON-документ корректной JSON Schema.

Входящие параметры:

- `json_data (JSONB)` - JSON Schema для проверки.

Выходные параметры:

- Возвращает `BINARY_INTEGER`: `1`, если схема корректна, иначе `0`.

Типичные сценарии использования:

- Предварительная проверка схемы перед использованием в `IS_VALID`.
- Контроль корректности пользовательских JSON Schema.

Примеры запросов:

```
DO $$
DECLARE
    v_res BINARY_INTEGER;
BEGIN
    v_res := dbms_json_schema_is_schema_valid('{"type":"string"}'::jsonb);

    RAISE NOTICE 'Прочитано: result=%', v_res;
    RAISE NOTICE 'DBMS_JSON_SCHEMA.IS_SCHEMA_VALID: УСПЕШНО';
END;
$$;
```

GET_VERSION

Функция возвращает версию реализации пакета `DBMS_JSON_SCHEMA` в RUDB.

Входящие параметры:

- У метода нет входящих параметров.

Выходные параметры:

- Возвращает `TEXT` с версией реализации.

Типичные сценарии использования:

- Диагностика версии пакета на тестовом или целевом стенде.
- Проверка совместимости между сборками RUDB.

Примеры запросов:

```
DO $$
DECLARE
    v_version TEXT;
BEGIN
    v_version := dbms_json_schema_get_version();

    RAISE NOTICE 'Версия: %', v_version;
    RAISE NOTICE 'DBMS_JSON_SCHEMA.GET_VERSION: УСПЕШНО';
END;
$$;
```

5.2.20. Пакет DBMS_FILE_TRANSFER

DBMS_FILE_TRANSFER_GET_VERSION

Возвращает версию реализации DBMS_FILE_TRANSFER.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_file_transfer_get_version() AS package_version;
```

DBMS_FILE_TRANSFER_GET_IS_SAFE_FILENAME

Проверяет, безопасно ли имя файла. Обычное имя файла ожидаемо safe=true. Имя с выходом в родительский каталог ожидаемо safe=false.

Входящие параметры:

- TEXT - имя файла.

Выходные параметры:

- BOOLEAN - true если безопасно, иначе false.

Типичные сценарии использования:

- Проверка безопасности имени файла, полученного от пользователя.

Примеры запросов:

```
dbms_file_transfer_is_safe_filename('rudb_file.txt') AS safe_filename;
dbms_file_transfer_is_safe_filename('../bad.txt') AS unsafe_filename;
```

DBMS_FILE_TRANSFER_COPY_FILE

Копирует server-side файл из одной директории в другую.

Входящие параметры:

- TEXT - директория исходного файла.
- TEXT - имя исходного файла.
- TEXT - директория итогового файла.
- TEXT - имя итогового файла.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Копирование server-side файла из одной директории в другую.

Примеры запросов:

```
DROP TABLE IF EXISTS dft_copy_src;
CREATE TEMP TABLE dft_copy_src(line text);

INSERT INTO dft_copy_src
VALUES ('hello from dbms_file_transfer_copy_file');

COPY dft_copy_src TO '/tmp/rudb_dft_copy_src.txt';

SELECT dbms_file_transfer_copy_file(
    '/tmp',
    'rudb_dft_copy_src.txt',
    '/tmp',
    'rudb_dft_copy_dst.txt'
);

SELECT pg_read_file('/tmp/rudb_dft_copy_dst.txt') AS copied_file_content;

SELECT pg_file_unlink_custom('/tmp/rudb_dft_copy_src.txt');
SELECT pg_file_unlink_custom('/tmp/rudb_dft_copy_dst.txt');
```

READ_FILE_CHUNK

Эта функция читает часть server-side файла.

Входящие параметры:

- TEXT - директория файла.
- TEXT - имя файла.
- INT8 - сдвиг по файлу.
- INT4 - размер части.

Выходные параметры:

- BYTEA - часть файла.

Типичные сценарии использования:

- Service-функция для чтения файла по частям.

Примеры запросов:

```
DROP TABLE IF EXISTS dft_read_chunk_src;

CREATE TEMP TABLE dft_read_chunk_src(line text);

INSERT INTO dft_read_chunk_src
VALUES ('hello from read_file_chunk');

COPY dft_read_chunk_src TO '/tmp/rudb_dft_read_chunk.txt';

SELECT encode(
    read_file_chunk('/tmp', 'rudb_dft_read_chunk.txt', 0, 64),
    'escape'
) AS file_chunk;

SELECT pg_file_unlink_custom('/tmp/rudb_dft_read_chunk.txt');
```

WRITE_FILE_CHUNK

Эта функция пишет bytea chunk в server-side файл.

Входящие параметры:

- TEXT - директория файла.
- TEXT - имя файла.

- BYTEA - часть файла.
- INT8 - сдвиг по файлу.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Service-функция для записи файла по частям.

Примеры запросов:

```
SELECT write_file_chunk(
    '/tmp',
    'rudb_dft_write_chunk.txt',
    convert_to('hello from write_file_chunk', 'UTF8'),
    0
);

SELECT pg_read_file('/tmp/rudb_dft_write_chunk.txt') AS written_file_content;

SELECT pg_file_unlink_custom('/tmp/rudb_dft_write_chunk.txt');
```

PG_FILE_UNLINK_CUSTOM

Эта функция удаляет server-side файл по полному пути.

Входящие параметры:

- TEXT - имя файла.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Service-функция для удаления файла.

Примеры запросов:

```
DROP TABLE IF EXISTS dft_unlink_src;

CREATE TEMP TABLE dft_unlink_src(line text);

INSERT INTO dft_unlink_src
VALUES ('file for pg_file_unlink_custom');

COPY dft_unlink_src TO '/tmp/rudb_dft_unlink.txt';

SELECT (pg_stat_file('/tmp/rudb_dft_unlink.txt', true)).size IS NOT NULL AS
file_exists_before;

SELECT pg_file_unlink_custom('/tmp/rudb_dft_unlink.txt');

SELECT (pg_stat_file('/tmp/rudb_dft_unlink.txt', true)).size IS NOT NULL AS
file_exists_after;
```

DBMS_FILE_TRANSFER_GET_FILE

Копирует файл с удалённой стороны через dblink.

Входящие параметры:

- TEXT - директория исходного файла.
- TEXT - имя исходного файла.
- TEXT - имя соединения.
- TEXT - директория итогового файла.
- TEXT - имя итогового файла.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Копирование файла с помощью dblink.

Примеры запросов:

```
CREATE EXTENSION IF NOT EXISTS dblink;

DROP TABLE IF EXISTS dft_get_src;

CREATE TEMP TABLE dft_get_src(line text);

INSERT INTO dft_get_src
VALUES ('hello from dbms_file_transfer_get_file');

COPY dft_get_src TO '/tmp/rudb_dft_get_remote_src.txt';

DO $$
BEGIN
    IF 'rudb_dft_get_link' = ANY (dblink_get_connections()) THEN
        PERFORM dblink_disconnect('rudb_dft_get_link');
    END IF;
END;
$$;

SELECT dblink_connect(
    'rudb_dft_get_link',
    'dbname=' || current_database() || ' user=' || current_user
);

SELECT dbms_file_transfer_get_file(
    '/tmp',
    'rudb_dft_get_remote_src.txt',
    'rudb_dft_get_link',
    '/tmp',
    'rudb_dft_get_local_dst.txt'
);

SELECT pg_read_file('/tmp/rudb_dft_get_local_dst.txt') AS copied_from_dblink;

SELECT dblink_disconnect('rudb_dft_get_link');

SELECT pg_file_unlink_custom('/tmp/rudb_dft_get_remote_src.txt');
SELECT pg_file_unlink_custom('/tmp/rudb_dft_get_local_dst.txt');
```

DBMS_FILE_TRANSFER_PUT_FILE

Копирует файл на удалённую сторону через dblink

Входящие параметры:

- TEXT - директория исходного файла.
- TEXT - имя исходного файла.
- TEXT - имя соединения.
- TEXT - директория итогового файла.
- TEXT - имя итогового файла.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Копирование файла на удалённый ресурс с помощью dblink.

Примеры запросов:

```
CREATE EXTENSION IF NOT EXISTS dblink;

DROP TABLE IF EXISTS dft_put_src;

CREATE TEMP TABLE dft_put_src(line text);

INSERT INTO dft_put_src
VALUES ('hello from dbms_file_transfer_put_file');

COPY dft_put_src TO '/tmp/rudb_dft_put_local_src.txt';

DO $$
BEGIN
    IF 'rudb_dft_put_link' = ANY (dblink_get_connections()) THEN
        PERFORM dblink_disconnect('rudb_dft_put_link');
    END IF;
END;
$$;

SELECT dblink_connect(
    'rudb_dft_put_link',
    'dbname=' || current_database() || ' user=' || current_user
);

SELECT dbms_file_transfer_put_file(
    '/tmp',
    'rudb_dft_put_local_src.txt',
    'rudb_dft_put_link',
    '/tmp',
    'rudb_dft_put_remote_dst.txt'
);

SELECT pg_read_file('/tmp/rudb_dft_put_remote_dst.txt') AS copied_to_dblink;

SELECT dblink_disconnect('rudb_dft_put_link');

SELECT pg_file_unlink_custom('/tmp/rudb_dft_put_local_src.txt');
SELECT pg_file_unlink_custom('/tmp/rudb_dft_put_remote_dst.txt');
```

5.2.21. Пакет DBMS_SYSTEM

DBMS_SYSTEM_GET_VERSION

Возвращает версию реализации DBMS_SYSTEM.

Входящие параметры:

- Отсутствуют.

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_system_get_version() AS package_version;
```

DBMS_SYSTEM_IS_VALID_SESSION

Проверяет, считается ли указанная сессия валидной для DBMS_SYSTEM.

Входящие параметры:

- sid INT4 - Идентификатор backend-сессии.
- serial_num INT4 - Совместимый аргумент SERIAL#.

Выходные параметры:

- BOOLEAN - `true`, если сессия доступна для DBMS_SYSTEM.

Типичные сценарии использования:

- Проверка валидности сессии для DBMS_SYSTEM.

Примеры запросов:

```
pg_backend_pid() AS current_backend_pid,
    dbms_system_is_valid_session(pg_backend_pid(), 1) AS current_session_is_valid;
```

DBMS_SYSTEM_KSDWRT

Пишет диагностическое сообщение в server log/trace.

Входящие параметры:

- dest INT4 - Назначение записи.
- message TEXT - Текст диагностического сообщения.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Отладка DBMS_SYSTEM.

Примеры запросов:

```
SELECT dbms_system_ksdwrt(
    2,
    'DBMS_SYSTEM.KSDWRT autonomous qsql example'
);
```

DBMS_SYSTEM_SET_EV

Устанавливает уровень event 10046 для текущей backend-сессии. Поддерживается только event `10046` для текущей backend-сессии. `level = 0` выключает SQL trace, `level > 0` включает trace.

Входящие параметры:

- sid INT4 - Идентификатор backend-сессии.
- serial INT4 - Совместимый аргумент SERIAL#.
- event_num INT4 - Номер диагностического event.
- level INT4 - Уровень trace.
- name TEXT - Диагностическое имя операции.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Отладка DBMS_SYSTEM.

Примеры запросов:

```
SELECT dbms_system_set_ev(pg_backend_pid(), 1, 10046, 0, '');

SELECT dbms_system_set_ev(pg_backend_pid(), 1, 10046, 1, '');

SELECT dbms_system_read_ev(10046) AS event_10046_level_after_set_ev;

SELECT dbms_system_set_ev(pg_backend_pid(), 1, 10046, 0, '');
```

DBMS_SYSTEM_READ_EV

Читает текущий уровень event. Для event `10046` возвращает локальный уровень SQL trace текущего backend-а. Для остальных events возвращает `0`.

Входящие параметры:

- event_num INT4 - Номер диагностического event.

Выходные параметры:

- INT4 - Текущий уровень event

Типичные сценарии использования:

- Отладка DBMS_SYSTEM.

Примеры запросов:

```
SELECT dbms_system_set_ev(pg_backend_pid(), 1, 10046, 1, '');

SELECT dbms_system_read_ev(10046) AS event_10046_level_enabled;

SELECT dbms_system_set_ev(pg_backend_pid(), 1, 10046, 0, '');

SELECT dbms_system_read_ev(10046) AS event_10046_level_disabled;
```

DBMS_SYSTEM_SET_SQL_TRACE_IN_SESSION

Oracle-like wrapper для включения/выключения SQL trace в сессии.

Входящие параметры:

- sid INT4 - Идентификатор backend-сессии.
- serial_num INT4 - Совместимый аргумент SERIAL#.

- sql_trace BOOLEAN - Признак включения SQL trace.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Отладка DBMS_SYSTEM.

Примеры запросов:

```
SELECT dbms_system_set_sql_trace_in_session(pg_backend_pid(), 1, true);

SELECT dbms_system_read_ev(10046) AS trace_level_after_enable;

SELECT 1 AS traced_query;

SELECT dbms_system_set_sql_trace_in_session(pg_backend_pid(), 1, false);

SELECT dbms_system_read_ev(10046) AS trace_level_after_disable;
```

5.2.22. Пакет DBMS_TNS

DBMS_TNS_GET_VERSION

Возвращает версию реализации DBMS_TNS.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_tns_get_version() AS package_version;
```

DBMS_TNS_HAS_TNSNAMES_CONFIG

Эта функция проверяет, доступен ли tnsnames.ora для текущей настройки dbms_tns.tns_admin.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- BOOLEAN - `true`, если конфигурация `tnsnames.ora` доступна.

Типичные сценарии использования:

- Сервис-функция для проверки доступа к tnsnames.ora.

Примеры запросов:

```
SET dbms_tns.tns_admin = '/tmp/rudb_tns_missing_dir';

SELECT dbms_tns_has_tnsnames_config() AS has_tnsnames_config;
```

DBMS_TNS_RESOLVE_TNSNAME

Разрешает Oracle Net имя подключения в connect descriptor.

Входящие параметры:

- TEXT - TNS alias или простая Easy Connect строку.

Выходные параметры:

- TEXT - resolved connect descriptor либо SQL NULL, если имя не может быть разрешено поддержанными источниками.

Типичные сценарии использования:

- Проверка поведения для неизвестного TNS alias.

Примеры запросов:

```
SET dbms_tns.tns_admin = '/tmp/rudb_tns_missing_dir';

SELECT dbms_tns_resolve_tnsname('UNKNOWN_ALIAS') IS NULL AS unknown_alias_is_null;
```

5.2.23. Пакет DBMS_WARNING

DBMS_WARNING_GET_VERSION

Возвращает версию реализации DBMS_WARNING.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_warning_get_version() AS package_version;
```

DBMS_WARNING_GET_WARNING_SETTING_STRING

Возвращает текущую строку warning settings.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- VARCHAR2 - Каноническая строка `PLSQL_WARNINGS`.

Типичные сценарии использования:

- Получение строки warning settings.

Примеры запросов:

```
SELECT dbms_warning_set_warning_setting_string('ENABLE:ALL', 'SESSION');

SELECT dbms_warning_get_warning_setting_string() AS warning_setting;
```

DBMS_WARNING_SET_WARNING_SETTING_STRING

Устанавливает warning settings для текущей сессии.

Входящие параметры:

- value VARCHAR2 - Oracle-like строка настройки предупреждений.
- scope VARCHAR2 - Область применения настройки. scope='SESSION' меняет настройку только в текущей сессии.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Установление строки warning settings.

Примеры запросов:

```
SELECT dbms_warning_set_warning_setting_string(
    'ENABLE:ALL,DISABLE:PERFORMANCE',
    'SESSION'
);
SELECT dbms_warning_get_warning_setting_string() AS warning_setting_after_set;
```

DBMS_WARNING_NORMALIZE_SETTING

Нормализует Oracle-like строку warning settings.

Входящие параметры:

- setting VARCHAR2 - Строковое представление настройки предупреждений.

Выходные параметры:

- VARCHAR2 - Каноническая строка настройки.

Типичные сценарии использования:

- Нормализация строки warning settings.

Примеры запросов:

```
SELECT dbms_warning_normalize_setting(
    ' Enable : ( 06002 , 06003 ) '
) AS normalized_setting;
```

DBMS_ADD_WARNING_SETTING_CAT

Добавляет/меняет настройку warning по категории.

Входящие параметры:

- warning_category VARCHAR2 - Категория предупреждения.
- warning_value VARCHAR2 - Значение настройки, например `ENABLE`, `DISABLE` или `ERROR`.
- scope VARCHAR2 - Область применения настройки.

Выходные параметры:

- Отсутствует.

Типичные сценарии использования:

- Изменение настроки warning.

Примеры запросов:

```
SELECT dbms_warning_set_warning_setting_string('DISABLE:ALL', 'SESSION');

SELECT dbms_warning_add_warning_setting_cat('SEVERE', 'ERROR', 'SESSION');

SELECT
    COALESCE(dbms_warning_get_warning_setting_cat('SEVERE'), '<NULL>') AS
severe_category_setting;
```

DBMS_GET_WARNING_SETTING_CAT

Возвращает настройку warning по категории.

Входящие параметры:

- warning_category VARCHAR2 - Категория предупреждения.

Выходные параметры:

- VARCHAR2 - Строковое значение настройки предупреждений.

Типичные сценарии использования:

- Изменение настроки warning.

Примеры запросов:

```
SELECT dbms_warning_set_warning_setting_string('DISABLE:ALL', 'SESSION');

SELECT dbms_warning_add_warning_setting_cat('SEVERE', 'ERROR', 'SESSION');

SELECT
    COALESCE(dbms_warning_get_warning_setting_cat('SEVERE'), '<NULL>') AS
severe_category_setting;
```

DBMS_WARNING_ADD_WARNING_SETTING_CAT

Добавляет/меняет настройку warning по номеру.

Входящие параметры:

- warning_number BINARY_INTEGER - Номер предупреждения.
- warning_value VARCHAR2 - Значение настройки, например `ENABLE`, `DISABLE` или `ERROR`.
- scope VARCHAR2 - Область применения настройки.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Изменение настроки warning.

Примеры запросов:

```
SELECT dbms_warning_set_warning_setting_string('DISABLE:ALL', 'SESSION');

SELECT dbms_warning_add_warning_setting_num(6002, 'ENABLE', 'SESSION');

SELECT
    COALESCE(dbms_warning_get_warning_setting_num(6002), '<NULL>') AS
warning_6002_setting;
```

DBMS_WARNING_GET_WARNING_SETTING_CAT

Возвращает настройку warning по номеру.

Входящие параметры:

- warning_number BINARY_INTEGER - Номер предупреждения.

Выходные параметры:

- VARCHAR2 - Строковое значение настройки предупреждений.

Типичные сценарии использования:

- Изменение настроки warning.

Примеры запросов:

```
SELECT dbms_warning_set_warning_setting_string('DISABLE:ALL', 'SESSION');

SELECT dbms_warning_add_warning_setting_num(6002, 'ENABLE', 'SESSION');

SELECT
    COALESCE(dbms_warning_get_warning_setting_num(6002), '<NULL>') AS
warning_6002_setting;
```

DBMS_WARNING_GET_CATEGORY

Возвращает категорию warning по номеру.

Входящие параметры:

- warning_number BINARY_INTEGER - Номер предупреждения.

Выходные параметры:

- VARCHAR2 - Строковое значение категории.

Типичные сценарии использования:

- Изменение настроки warning.

Примеры запросов:

```
SELECT
    COALESCE(dbms_warning_get_category(6002), '<NULL>') AS warning_6002_category;
```

5.2.24. Пакет DBMS_TRACE

DBMS_TRACE_GET_VERSION

Возвращает версию реализации DBMS_TRACE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_trace_get_version() AS package_version;
```

DBMS_TRACE_IS_ENABLED

Показывает, включена ли DBMS_TRACE-трассировка в текущей сессии.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- BOOLEAN - `true`, если трассировка включена.

Типичные сценарии использования:

- Получение информации о DBMS_TRACE.

Примеры запросов:

```
SELECT dbms_trace_clear_plsql_trace();

SELECT dbms_trace_is_enabled() AS trace_enabled_before;

SELECT dbms_trace_set_plsql_trace(1);

SELECT dbms_trace_is_enabled() AS trace_enabled_after;

SELECT dbms_trace_clear_plsql_trace();
```

DBMS_TRACE_SET_PLSQL_TRACE

Включает session-local PL/pgSQL trace. После вызова dbms_trace_is_enabled() должен вернуть true.

Входящие параметры:

- trace_level INT4 - Уровень или маска трассировки.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Включение session-local PL/pgSQL trace.

Примеры запросов:

```
SELECT dbms_trace_clear_plsql_trace();

SELECT dbms_trace_set_plsql_trace(1);

SELECT dbms_trace_is_enabled() AS trace_enabled;

SELECT dbms_trace_clear_plsql_trace();
```

DBMS_TRACE_CLEAR_PLSQL_TRACE

Выключает session-local PL/pgSQL trace.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Выключение session-local PL/pgSQL trace.

Примеры запросов:

```
SELECT dbms_trace_set_plsql_trace(1);

SELECT dbms_trace_is_enabled() AS trace_enabled_before_clear;

SELECT dbms_trace_clear_plsql_trace();

SELECT dbms_trace_is_enabled() AS trace_enabled_after_clear;
```

DBMS_TRACE_PLSQL_TRACE_RUNS

Показывает trace runs текущей сессии.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- RECORD - Набор строк trace runs.

Типичные сценарии использования:

- Показ session-local PL/pgSQL trace.

Примеры запросов:

```
DROP FUNCTION IF EXISTS dbms_trace_runs_demo(integer);

CREATE FUNCTION dbms_trace_runs_demo(x integer)
RETURNS integer
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN x + 1;
END;
$$;

SELECT dbms_trace_clear_plsql_trace();
SELECT dbms_trace_set_plsql_trace(1);

SELECT dbms_trace_runs_demo(10) AS traced_result;

SELECT
    run_id,
    trace_mask,
    is_active,
    event_count
FROM dbms_trace_plsql_trace_runs() AS r(
    run_id integer,
    trace_mask integer,
    is_active boolean,
    event_count integer
)
ORDER BY run_id;

SELECT dbms_trace_clear_plsql_trace();

DROP FUNCTION dbms_trace_runs_demo(integer);
```

DBMS_TRACE_PLSQL_TRACE_EVENTS

Показывает накопленные PL/pgSQL trace events текущей сессии.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- RECORD - Набор строк trace events.

Типичные сценарии использования:

- Показ session-local PL/pgSQL trace events.

Примеры запросов:

```
DROP FUNCTION IF EXISTS dbms_trace_events_outer(integer);
DROP FUNCTION IF EXISTS dbms_trace_events_inner(integer);

CREATE FUNCTION dbms_trace_events_inner(x integer)
RETURNS integer
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN x + 1;
END;
$$;

CREATE FUNCTION dbms_trace_events_outer(x integer)
RETURNS integer
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN dbms_trace_events_inner(x) + 1;
END;
$$;

SELECT dbms_trace_clear_plsql_trace();
SELECT dbms_trace_set_plsql_trace(1);

SELECT dbms_trace_events_outer(10) AS traced_result;

SELECT
    event_type,
    run_id,
    event_seq,
    unit_name,
    subprogram_name
FROM dbms_trace_plsql_trace_events() AS e(
    event_type text,
    run_id integer,
    event_seq integer,
    unit_name text,
    subprogram_name text
)
ORDER BY run_id, event_seq;

SELECT dbms_trace_clear_plsql_trace();

DROP FUNCTION dbms_trace_events_outer(integer);
DROP FUNCTION dbms_trace_events_inner(integer);
```

5.2.25. Пакет DBMS_DB_VERSION

DBMS_DB_VERSION_GET_VERSION

Возвращает версию реализации DBMS_DB_VERSION.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_get_version() AS package_version;
```

DBMS_DB_VERSION_VERSION

озвращает номер VERSION пакета DBMS_DB_VERSION.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - номер VERSION пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_version() AS version_value;
```

DBMS_DB_VERSION_RELEASE

Возвращает номер RELEASE пакета DBMS_DB_VERSION.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - номер RELEASE пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_release() AS release_value;
```

DBMS_DB_VERSION_VER_LE

Проверяет, меньше или равна текущая compatibility version заданному порогу.

Входящие параметры:

- version INT4 - Проверяемое значение VERSION.
- release INT4 - Проверяемое значение RELEASE.

Выходные параметры:

- BOOLEAN - `true`, если compatibility version пакета не выше указанной.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le(23, 0) AS ver_le_23_0;
```

DBMS_DB_VERSION_VER_LE_9

Возвращает флаг DBMS_DB_VERSION.VER_LE_9.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_9.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_9() AS ver_le_9;
```

DBMS_DB_VERSION_VER_LE_9_1

Возвращает флаг DBMS_DB_VERSION.VER_LE_9_1.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_9_1.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_9_1() AS ver_le_9_1;
```

DBMS_DB_VERSION_VER_LE_9_2

Возвращает флаг DBMS_DB_VERSION.VER_LE_9_2.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_9_2.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_9_2() AS ver_le_9_2;
```

DBMS_DB_VERSION_VER_LE_10

Возвращает флаг DBMS_DB_VERSION.VER_LE_10.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_10.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_10() AS ver_le_10;
```

DBMS_DB_VERSION_VER_LE_10_1

Возвращает флаг DBMS_DB_VERSION.VER_LE_10_1.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_10_1.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_10_1() AS ver_le_10_1;
```

DBMS_DB_VERSION_VER_LE_10_2

Возвращает флаг DBMS_DB_VERSION.VER_LE_10_2.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_10_2.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_10_2() AS ver_le_10_2;
```

DBMS_DB_VERSION_VER_LE_11

Возвращает флаг DBMS_DB_VERSION.VER_LE_11.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_11.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_11() AS ver_le_11;
```

DBMS_DB_VERSION_VER_LE_11_1

Возвращает флаг DBMS_DB_VERSION.VER_LE_11_1.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_11_1.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_11_1() AS ver_le_11_1;
```

DBMS_DB_VERSION_VER_LE_11_2

Возвращает флаг DBMS_DB_VERSION.VER_LE_11_2.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_11_2.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_11_2() AS ver_le_11_2;
```

DBMS_DB_VERSION_VER_LE_12

Возвращает флаг DBMS_DB_VERSION.VER_LE_12.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_12.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_12() AS ver_le_12;
```

DBMS_DB_VERSION_VER_LE_12_1

Возвращает флаг DBMS_DB_VERSION.VER_LE_12_1.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_12_1.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_12_1() AS ver_le_12_1;
```

DBMS_DB_VERSION_VER_LE_12_2

Возвращает флаг DBMS_DB_VERSION.VER_LE_12_2.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_12_2.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_12_2() AS ver_le_12_2;
```

DBMS_DB_VERSION_VER_LE_18

Возвращает флаг DBMS_DB_VERSION.VER_LE_18.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_18.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_18() AS ver_le_18;
```

DBMS_DB_VERSION_VER_LE_19

Возвращает флаг DBMS_DB_VERSION.VER_LE_19.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_19.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_19() AS ver_le_19;
```

DBMS_DB_VERSION_VER_LE_20

Возвращает флаг DBMS_DB_VERSION.VER_LE_20.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_20.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_20() AS ver_le_20;
```

DBMS_DB_VERSION_VER_LE_21

Возвращает флаг DBMS_DB_VERSION.VER_LE_21.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_21.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_21() AS ver_le_21;
```

DBMS_DB_VERSION_VER_LE_23

Возвращает флаг DBMS_DB_VERSION.VER_LE_23.

Входящие параметры:

- Отсутствует

Выходные параметры:

- BOOLEAN - флаг DBMS_DB_VERSION.VER_LE_23.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_db_version_ver_le_23() AS ver_le_23;
```

5.2.26. Пакет DBMS_STAT_FUNCS

DBMS_STAT_FUNCS_GET_VERSION

Возвращает версию реализации DBMS_STAT_FUNCS.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_stat_funcs_get_version() AS package_version;
```

DBMS_STAT_FUNCS_KS_PVALUE

Вычисляет p-value для статистики Колмогорова-Смирнова.

Входящие параметры:

- statistic FLOAT8 - Значение статистики критерия.
- sample_count INT4 - Объём выборки.

Выходные параметры:

- FLOAT8 - Значимость критерия либо SQL `NULL` для неподдержанного ввода.

Типичные сценарии использования:

- Статистические вычисления.

Примеры запросов:

```
SELECT dbms_stat_funcs_ks_pvalue(0.1::double precision, 20) AS ks_pvalue;
```

DBMS_STAT_FUNCS_SUMMARY

Возвращает сводную статистику по числовой колонке.

Входящие параметры:

- ownername TEXT - Имя владельца таблицы
- tablename TEXT - Имя таблицы.
- columnname TEXT - Имя числовой колонки
- sigma_value NUMERIC - Числовое количество сигм для границ и

экстремальных значений.

Выходные параметры:

- RECORD - Именованный record со сводной статистикой

Типичные сценарии использования:

- Статистические вычисления.

Примеры запросов:

```
DROP TABLE IF EXISTS dbms_stat_funcs_summary_data;

CREATE TEMP TABLE dbms_stat_funcs_summary_data(x numeric);

INSERT INTO dbms_stat_funcs_summary_data(x)
VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

SELECT *
FROM dbms_stat_funcs_summary(
    (pg_my_temp_schema()::regnamespace)::text,
    'dbms_stat_funcs_summary_data',
    'x',
    2::numeric
);
```

DBMS_STAT_FUNCS_EXPONENTIAL_DIST_FIT

Подбор экспоненциального распределения.

Входящие параметры:

- ownername TEXT - Имя владельца таблицы
- tablename TEXT - Имя таблицы.
- columnname TEXT - Имя числовой колонки
- test_type TEXT - Тип проверки: `KOLMOGOROV_SMIRNOV`, `ANDERSON_DARLING`, `CHI_SQUARED`
- lambda NUMERIC - Параметр интенсивности распределения.
- mu NUMERIC - Параметр сдвига распределения.

Выходные параметры:

- RECORD - Именованный record с полями `lambda`, `mu`, `sig`.

Типичные сценарии использования:

- Статистические вычисления.

Примеры запросов:

```
DROP TABLE IF EXISTS dbms_stat_funcs_exp_data;

CREATE TEMP TABLE dbms_stat_funcs_exp_data(x numeric);

INSERT INTO dbms_stat_funcs_exp_data(x)
VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

SELECT *
FROM dbms_stat_funcs_exponential_dist_fit(
    (pg_my_temp_schema()::regnamespace)::text,
    'dbms_stat_funcs_exp_data',
    'x',
    'KOLMOGOROV_SMIRNOV',
    NULL::numeric,
    NULL::numeric
);
```

DBMS_STAT_FUNCS_NORMAL_DIST_FIT

Подбор нормального распределения.

Входящие параметры:

- ownername TEXT - Имя владельца таблицы
- tablename TEXT - Имя таблицы.
- columnname TEXT - Имя числовой колонки
- test_type TEXT - Тип проверки: `SHAPIRO_WILKS`, `KOLMOGOROV_SMIRNOV`, `ANDERSON_DARLING`, `CHI_SQUARED`.
- mean NUMERIC - Параметр среднего значения.
- stdev NUMERIC - Параметр стандартного отклонения.

Выходные параметры:

- RECORD - Именованный record с полями `mean`, `stdev`, `sig`.

Типичные сценарии использования:

- Статистические вычисления.

Примеры запросов:

```
DROP TABLE IF EXISTS dbms_stat_funcs_normal_data;

CREATE TEMP TABLE dbms_stat_funcs_normal_data(x numeric);

INSERT INTO dbms_stat_funcs_normal_data(x)
VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

SELECT *
FROM dbms_stat_funcs_normal_dist_fit(
    (pg_my_temp_schema()::regnamespace)::text,
    'dbms_stat_funcs_normal_data',
    'x',
    'KOLMOGOROV_SMIRNOV',
    NULL::numeric,
    NULL::numeric
);
```

DBMS_STAT_FUNCS_POISSON_DIST_FIT

Подбор распределения Пуассона.

Входящие параметры:

- ownername TEXT - Имя владельца таблицы
- tablename TEXT - Имя таблицы.
- columnname TEXT - Имя числовой колонки
- test_type TEXT - Тип проверки: `KOLMOGOROV_SMIRNOV`, `ANDERSON_DARLING`, `CHI_SQUARED`.
- lambda NUMERIC - Интенсивность распределения Пуассона.

Выходные параметры:

- RECORD - Именованный record с полями `lambda`, `sig`.

Типичные сценарии использования:

- Статистические вычисления.

Примеры запросов:

```
DROP TABLE IF EXISTS dbms_stat_funcs_poisson_data;

CREATE TEMP TABLE dbms_stat_funcs_poisson_data(x numeric);

INSERT INTO dbms_stat_funcs_poisson_data(x)
VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

SELECT *
FROM dbms_stat_funcs_poisson_dist_fit(
    (pg_my_temp_schema()::regnamespace)::text,
    'dbms_stat_funcs_poisson_data',
    'x',
    'KOLMOGOROV_SMIRNOV',
    NULL::numeric
);
```

DBMS_STAT_FUNCS_UNIFORM_DIST_FIT

Подбор равномерного распределения.

Входящие параметры:

- ownername TEXT - Имя владельца таблицы
- tablename TEXT - Имя таблицы.
- columnname TEXT - Имя числовой колонки
- var_type TEXT - Тип распределения: `CONTINUOUS`, `DISCRETE`.
- test_type TEXT - Тип проверки: `KOLMOGOROV_SMIRNOV`, `ANDERSON_DARLING`, `CHI_SQUARED`.
- parama NUMERIC - Нижняя граница распределения.
- paramb NUMERIC - Верхняя граница распределения.

Выходные параметры:

- RECORD - Именованный record с полями `parama`, `paramb`, `sig`.

Типичные сценарии использования:

- Статистические вычисления.

Примеры запросов:

```
DROP TABLE IF EXISTS dbms_stat_funcs_uniform_data;

CREATE TEMP TABLE dbms_stat_funcs_uniform_data(x numeric);

INSERT INTO dbms_stat_funcs_uniform_data(x)
VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

SELECT *
FROM dbms_stat_funcs_uniform_dist_fit(
    (pg_my_temp_schema()::regnamespace)::text,
    'dbms_stat_funcs_uniform_data',
    'x',
    NULL::text,
    'KOLMOGOROV_SMIRNOV',
    NULL::numeric,
    NULL::numeric
);
```

DBMS_STAT_FUNCS_WEIBULL_DIST_FIT

Подбор распределения Вейбулла.

Входящие параметры:

- ownername TEXT - Имя владельца таблицы
- tablename TEXT - Имя таблицы.
- columnname TEXT - Имя числовой колонки
- test_type TEXT - Тип проверки: `KOLMOGOROV_SMIRNOV`, `ANDERSON_DARLING`, `CHI_SQUARED`.
- alpha NUMERIC - Параметр масштаба распределения.
- mu NUMERIC - Параметр сдвига распределения.
- beta NUMERIC - Параметр формы распределения.

Выходные параметры:

- RECORD - Именованный record с полями `alpha`, `mu`, `beta`, `sig`.

Типичные сценарии использования:

- Статистические вычисления.

Примеры запросов:

```
DROP TABLE IF EXISTS dbms_stat_funcs_weibull_data;

CREATE TEMP TABLE dbms_stat_funcs_weibull_data(x numeric);

INSERT INTO dbms_stat_funcs_weibull_data(x)
VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

SELECT *
FROM dbms_stat_funcs_weibull_dist_fit(
    (pg_my_temp_schema()::regnamespace)::text,
    'dbms_stat_funcs_weibull_data',
    'x',
    'KOLMOGOROV_SMIRNOV',
    NULL::numeric,
    NULL::numeric,
    NULL::numeric
);
```

5.2.27. Пакет DBMS_SQLSET

DBMS_SQLSET_GET_VERSION

Возвращает версию реализации DBMS_SQLSET.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT dbms_sqlset_get_version() AS package_version;
```

DBMS_SQLSET_MODE_EPLACE_OLD_STATS

Возвращает Oracle-совместимую константу

DBMS_SQLSET.MODE_REPLACE_OLD_STATS

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.MODE_REPLACE_OLD_STATS.

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_mode_replace_old_stats() AS mode_replace_old_stats;
```

DBMS_SQLSET_MODE_ACCUMULATE_STATS

Возвращает Oracle-совместимую константу DBMS_SQLSET.MODE_ACCUMULATE_STATS

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.MODE_ACCUMULATE_STATS

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_mode_accumulate_stats() AS mode_accumulate_stats;
```

DBMS_SQLSET_HAS_RECURSIVE_SQL

Возвращает Oracle-совместимую константу DBMS_SQLSET.HAS_RECURSIVE_SQL

Входящие параметры:

- Отсутствуют

Выходные параметры:

- VARCHAR2 - константа DBMS_SQLSET.HAS_RECURSIVE_SQL

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_has_recursive_sql() AS has_recursive_sql;
```

DBMS_SQLSET_NO_RECURSIVE_SQL

Возвращает Oracle-совместимую константу DBMS_SQLSET.NO_RECURSIVE_SQL

Входящие параметры:

- Отсутствуют

Выходные параметры:

- VARCHAR2 - константа DBMS_SQLSET.NO_RECURSIVE_SQL

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_no_recursive_sql() AS no_recursive_sql;
```

DBMS_SQLSET_SINGLE_EXECUTION

Возвращает Oracle-совместимую константу DBMS_SQLSET.SINGLE_EXECUTION

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.SINGLE_EXECUTION

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_single_execution() AS single_execution;
```

DBMS_SQLSET_ALL_EXECUTIONS

Возвращает Oracle-совместимую константу DBMS_SQLSET.ALL_EXECUTIONS

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.ALL_EXECUTIONS

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_all_executions() AS all_executions;
```

DBMS_SQLSET_LIMITED_COMMAND_TYPE

Возвращает Oracle-совместимую константу DBMS_SQLSET.LIMITED_COMMAND_TYPE

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.LIMITED_COMMAND_TYPE

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_limited_command_type() AS limited_command_type;
```

DBMS_SQLSET_ALL_COMMAND_TYPE

Возвращает Oracle-совместимую константу DBMS_SQLSET.ALL_COMMAND_TYPE

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.ALL_COMMAND_TYPE

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_all_command_type() AS all_command_type;
```

DBMS_SQLSET_STS_STGTAB_10_2_VERSION

Возвращает Oracle-совместимую константу DBMS_SQLSET.STS_STGTAB_10_2_VERSION

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.STS_STGTAB_10_2_VERSION

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_sts_stgtab_10_2_version() AS sts_stgtab_10_2_version;
```

DBMS_SQLSET_STS_STGTAB_11_1_VERSION

Возвращает Oracle-совместимую константу DBMS_SQLSET.STS_STGTAB_11_1_VERSION

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.STS_STGTAB_11_1_VERSION

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_sts_stgtab_11_1_version() AS sts_stgtab_11_1_version;
```

DBMS_SQLSET_STS_STGTAB_11_2_VERSION

Возвращает Oracle-совместимую константу DBMS_SQLSET.STS_STGTAB_11_2_VERSION

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.STS_STGTAB_11_2_VERSION

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_sts_stgtab_11_2_version() AS sts_stgtab_11_2_version;
```

DBMS_SQLSET_STS_STGTAB_12_1_VERSION

Возвращает Oracle-совместимую константу DBMS_SQLSET.STS_STGTAB_12_1_VERSION

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.STS_STGTAB_12_1_VERSION

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_sts_stgtab_12_1_version() AS sts_stgtab_12_1_version;
```

DBMS_SQLSET_STS_STGTAB_12_2_VERSION

Возвращает Oracle-совместимую константу DBMS_SQLSET.STS_STGTAB_12_2_VERSION

Входящие параметры:

- Отсутствуют

Выходные параметры:

- INT4 - константа DBMS_SQLSET.STS_STGTAB_12_2_VERSION

Типичные сценарии использования:

- Просмотр констант DBMS_SQLSET

Примеры запросов:

```
SELECT dbms_sqlset_sts_stgtab_12_2_version() AS sts_stgtab_12_2_version;
```

DBMS_SQLSET_NORMALIZE_ATTRIBUTE_LIST

Нормализует attribute_list

Входящие параметры:

- attribute_list TEXT - Имя режима атрибутов.

Выходные параметры:

- TEXT - Нормализованное имя режима атрибутов.

Типичные сценарии использования:

- Нормализация списка атрибутов.

Примеры запросов:

```
SELECT dbms_sqlset_normalize_attribute_list('basic') AS normalized_basic;

SELECT dbms_sqlset_normalize_attribute_list('typical') AS normalized_typical;
```

DBMS_SQLSET_CREATE_SQLSET_0

Создаёт session-local SQL tuning set и возвращает его имя.

Входящие параметры:

- sqlset_name TEXT - Имя SQL tuning set или `NULL` для автогенерации.
- description TEXT - Описание SQL tuning set.
- sqlset_owner TEXT - Владелец набора; поддержан только текущий пользователь.

Выходные параметры:

- VARCHAR2 - Имя созданного SQL tuning set.

Типичные сценарии использования:

- Создание SQL tuning set.

Примеры запросов:

```
DO $$
BEGIN
    CALL dbms_sqlset_drop_sqlset('sts_create_func_demo', NULL);
EXCEPTION WHEN OTHERS THEN
    NULL;
END;
$$;

SELECT dbms_sqlset_create_sqlset_0(
    'sts_create_func_demo',
    'create function form demo',
    NULL
) AS created_sqlset_name;

CALL dbms_sqlset_drop_sqlset('sts_create_func_demo', NULL);
```

DBMS_SQLSET_CREATE_SQLSET_1

Создаёт session-local SQL tuning set без возвращаемого значения.

Входящие параметры:

- sqlset_name TEXT - Имя SQL tuning set или `NULL` для автогенерации.
- description TEXT - Описание SQL tuning set.
- sqlset_owner TEXT - Владелец набора; поддержан только текущий пользователь.

Выходные параметры:

- Отсутствует.

Типичные сценарии использования:

- Создание SQL tuning set.

Примеры запросов:

```
DO $$
BEGIN
    CALL dbms_sqlset_drop_sqlset('sts_create_proc_demo', NULL);
EXCEPTION WHEN OTHERS THEN
    NULL;
END;
$$;

CALL dbms_sqlset_create_sqlset_1(
    'sts_create_proc_demo',
    'create procedure form demo',
    NULL
);

SELECT
    count(*) AS rows_in_created_sqlset
FROM dbms_sqlset_select_sqlset(
    'sts_create_proc_demo',
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    1::numeric,
    10::numeric,
    'BASIC',
    NULL,
    NULL,
    NULL
);

CALL dbms_sqlset_drop_sqlset('sts_create_proc_demo', NULL);
```

DBMS_SQLSET_SELECT_SQLSET

Возвращает строки SQL tuning set.

Входящие параметры:

- sqlset_name TEXT - Имя SQL tuning set.
- basic_filter TEXT - SQLSET_ROW predicate; не поддерживается, кроме NULL/пустой строки.
- object_filter TEXT - Object-list filter; не поддерживается, кроме NULL/пустой строки.

- ranking_measure1 TEXT - Ranking expression; не поддерживается, кроме `NULL`/пустой строки.
- ranking_measure2 TEXT - Ranking expression; не поддерживается, кроме `NULL`/пустой строки.
- ranking_measure3 TEXT - Ranking expression; не поддерживается, кроме `NULL`/пустой строки.
- result_percentage NUMERIC - Поддержан только `NULL` или `1`.
- result_limit NUMERIC - Максимальное количество строк: `NULL`, `0` или положительное целое.
- attribute_list TEXT - Поддержаны `NULL`, `BASIC`, `TYPICAL`; `ALL` не поддерживается.
- plan_filter TEXT - Plan filter; не поддерживается, кроме `NULL`/пустой строки.
- sqlset_owner TEXT - Владелец набора; поддержан только текущий пользователь.
- recursive_sql TEXT - Поддержаны `NULL`, пустая строка, `HAS_RECURSIVE_SQL` и `Y`.

Выходные параметры:

- RECORD - Набор строк именованного record с поддерживаемым подмножеством SQLSET_ROW.

Типичные сценарии использования:

- Создание SQL tuning set.

Примеры запросов:

```
DO $$
BEGIN
    CALL dbms_sqlset_drop_sqlset('sts_select_empty_demo', NULL);
EXCEPTION WHEN OTHERS THEN
    NULL;
END;
$$;

SELECT dbms_sqlset_create_sqlset_0(
    'sts_select_empty_demo',
    'empty select demo',
    NULL
) AS created_sqlset_name;

SELECT
    count(*) AS selected_rows
FROM dbms_sqlset_select_sqlset(
    'sts_select_empty_demo',
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    1::numeric,
    10::numeric,
    'BASIC',
    NULL,
    NULL,
    NULL
);

CALL dbms_sqlset_drop_sqlset('sts_select_empty_demo', NULL);
```

DBMS_SQLSET_LOAD_SQLSET

Загружает строки SQLSET_ROW из refcursor в session-local SQL tuning set.

Входящие параметры:

- sqlset_name TEXT - Имя SQL tuning set.
- populate_cursor REFCURSOR - Имя открытого `refcursor` с SQLSET_ROW-compatible данными.
- load_option TEXT - Режим загрузки: `INSERT`, `UPDATE` или `MERGE`.

- update_option TEXT - Режим обновления: `REPLACE` или `ACCUMULATE`.
- update_condition TEXT - Условие обновления; не поддерживается, кроме `NULL`/пустой строки.
- update_attributes TEXT - Список обновляемых атрибутов; не поддерживается, кроме `NULL`/пустой строки.
- ignore_null BOOLEAN - Если `true`, входящий `NULL` не затирает существующее значение.
- commit_rows NUMERIC - Oracle-style параметр внутренних commit; поддержан только `NULL`.
- sqlset_owner TEXT - Владелец набора; поддержан только текущий пользователь.

Выходные параметры:

- Отсутствует.

Типичные сценарии использования:

- Создание SQL tuning set.

Примеры запросов:

```
DO $$
BEGIN
    CALL dbms_sqlset_drop_sqlset('sts_load_demo', NULL);
EXCEPTION WHEN OTHERS THEN
    NULL;
END;
$$;

SELECT dbms_sqlset_create_sqlset_0(
    'sts_load_demo',
    'LOAD_SQLSET test example',
    NULL
) AS created_sqlset_name;

DO $$
DECLARE
    c refcursor;
BEGIN
    OPEN c FOR
        SELECT *
```

```
        FROM (VALUES
            (
                'sql_load_1'::text,
                1001::numeric,
                'select 1'::text,
                current_schema()::text,
                'demo_module'::text,
                'demo_action_1'::text,
                10::numeric,
                20::numeric,
                30::numeric,
                40::numeric,
                0::numeric,
                50::numeric,
                1::numeric,
                2::numeric,
                2::numeric,
                100::numeric,
                NULL::bytea,
                1::numeric,
                3::numeric,
                NULL::text,
                60::numeric,
                60::numeric,
                NULL::text,
                111::numeric,
                NULL::numeric,
                NULL::text,
                0::numeric,
                1001::numeric,
                0::numeric,
                NULL::text,
                NULL::text,
                NULL::text,
                0::numeric,
                0::numeric,
                0::numeric,
                current_user::text
            ),
            (
                'sql_load_2'::text,
                1002::numeric,
                'select current_database()'::text,
                current_schema()::text,
                'demo_module'::text,
                'demo_action_2'::text,
                11::numeric,
```

```
            21::numeric,
            31::numeric,
            41::numeric,
            0::numeric,
            51::numeric,
            1::numeric,
            3::numeric,
            3::numeric,
            101::numeric,
            NULL::bytea,
            1::numeric,
            3::numeric,
            NULL::text,
            60::numeric,
            60::numeric,
            NULL::text,
            222::numeric,
            NULL::numeric,
            NULL::text,
            0::numeric,
            1002::numeric,
            0::numeric,
            NULL::text,
            NULL::text,
            NULL::text,
            0::numeric,
            0::numeric,
            0::numeric,
            current_user::text
        )
    ) AS v(
        sql_id,
        force_matching_signature,
        sql_text,
        parsing_schema_name,
        module,
        action,
        elapsed_time,
        cpu_time,
        buffer_gets,
        disk_reads,
        direct_writes,
        rows_processed,
        fetches,
        executions,
        end_of_fetch_count,
        optimizer_cost,
```

```
            optimizer_env,
            priority,
            command_type,
            first_load_time,
            stat_period,
            active_stat_period,
            other,
            plan_hash_value,
            con_dbid,
            last_exec_start_time,
            sharable_mem,
            exact_matching_signature,
            result_cache_executions,
            sql_profile,
            sql_plan_baseline,
            sql_patch,
            avg_hard_parse_time,
            user_io_wait_time,
            io_interconnect_bytes,
            parsing_user_name
        );

    CALL dbms_sqlset_load_sqlset(
        'sts_load_demo',
        c,
        'INSERT',
        'REPLACE',
        NULL,
        NULL,
        true,
        NULL,
        NULL
    );

    CLOSE c;
END;
$$;

SELECT count(*) AS loaded_rows
FROM dbms_sqlset_select_sqlset(
    'sts_load_demo',
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    1::numeric,
```

```
    10::numeric,
    'BASIC',
    NULL,
    NULL,
    NULL
);

SELECT
    sql_id,
    sql_text,
    parsing_schema_name,
    module,
    action
FROM dbms_sqlset_select_sqlset(
    'sts_load_demo',
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    1::numeric,
    10::numeric,
    'BASIC',
    NULL,
    NULL,
    NULL
)
ORDER BY sql_id;

CALL dbms_sqlset_drop_sqlset('sts_load_demo', NULL);
```

DBMS_SQLSET_DROP_SQLSET

Удаляет session-local SQL tuning set текущего пользователя

Входящие параметры:

- sqlset_name TEXT - Имя SQL tuning set.
- sqlset_owner TEXT - Владелец набора; поддержан только текущий пользователь.

Выходные параметры:

- Отсутствует.

Типичные сценарии использования:

- Удаление SQL tuning set.

Примеры запросов:

```
DO $$
BEGIN
    CALL dbms_sqlset_drop_sqlset('sts_drop_demo', NULL);
EXCEPTION WHEN OTHERS THEN
    NULL;
END;
$$;

SELECT dbms_sqlset_create_sqlset_0(
    'sts_drop_demo',
    'drop sqlset demo',
    NULL
) AS created_sqlset_name;

CALL dbms_sqlset_drop_sqlset('sts_drop_demo', NULL);

DO $$
BEGIN
    PERFORM *
    FROM dbms_sqlset_select_sqlset(
        'sts_drop_demo',
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        1::numeric,
        10::numeric,
        'BASIC',
        NULL,
        NULL,
        NULL
    );

    RAISE NOTICE 'unexpected: sqlset still exists';
EXCEPTION WHEN OTHERS THEN
    RAISE NOTICE 'expected: sqlset was dropped';
END;
$$;
```

5.2.28. Пакет DBMS_DDL

DBMS_DDL_CREATE_WRAPPED

Эта процедура создает обертку для указанного SQL DDL, позволяя выполнять его в дальнейшем.

Входящие параметры:

- ddl (text): SQL DDL для создания упаковки.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Создание обёрток для DDL команд.

Примеры запросов:

```
CALL dbms_ddl_create_wrapped('CREATE OR REPLACE FUNCTION test_func() RETURNS void AS
$$ BEGIN NULL; END; $$ LANGUAGE plpgsql;');
```

DBMS_DDL_WRAP

Эта функция создает обертку для указанного SQL DDL, позволяя его дальнейшее использование и выполнение.

Входящие параметры:

- ddl (text или bytea): SQL DDL, который необходимо завернуть.

Выходные параметры:

- text: Обернутый SQL DDL.

Типичные сценарии использования:

- Создание обёрток для DDL команд.

Примеры запросов:

```
SELECT dbms_ddl_wrap('CREATE OR REPLACE FUNCTION test_func() RETURNS void AS $$ BEGIN
NULL; END; $$ LANGUAGE plpgsql;');
```

5.2.29. Пакет DBMS_XMLSAVE

DBMS_XMLSAVE_NEWCONTEXT

Эта функция создает контекст сохранения и возвращает дескриптор контекста.

Входящие параметры:

- targettable (VARCHAR2) — имя целевой таблицы, для которой создается контекст сохранения XML.

Выходные параметры:

- INT4 - дескриптор контекста, который будет использоваться в последующих вызовах функций пакета.

Типичные сценарии использования:

- Преобразование XML-данных в типы баз данных.

Примеры запросов:

```
SELECT dbms_xmlsave_newcontext('my_target_table');
```

DBMS_XMLSAVE_INSERTXML

Вставляет XML-документ в таблицу, указанную при создании контекста, и возвращает количество вставленных строк. Параметры описаны в следующей таблице.

Входящие параметры:

- ctxhdl (INT4) — дескриптор контекста, полученный ранее с помощью функции dbms_xmlsave_newcontext.
- xdoc (VARCHAR2 или XMLTYPE) — XML-документ, содержащий данные для вставки.

Выходные параметры:

- NUMBER - количество успешно вставленных строк.

Типичные сценарии использования:

- Преобразование XML-данных в типы баз данных.

Примеры запросов:

```
SELECT dbms_xmlsave_insertxml(
    0,
    '<ROWSET><ROW><ID>1</ID><NAME>Test</NAME></ROW></ROWSET>'
);
```

DBMS_XMLSAVE_UPDATEXML

Обновляет указанную при создании контекста таблицу данными из XML-документа и возвращает количество обновленных строк.

Входящие параметры:

- ctxhdl (INT4) — дескриптор контекста, полученный ранее с помощью функции dbms_xmlsave_newcontext.
- xdoc (VARCHAR2 или XMLTYPE) — XML-документ, содержащий данные для обновления.

Выходные параметры:

- INT4 - количество успешно обновленных строк.

Типичные сценарии использования:

- Преобразование XML-данных в типы баз данных.

Примеры запросов:

```
SELECT dbms_xmlsave_updatexml(
    0,
    '<ROWSET><ROW><ID>1</ID><NAME>Updated</NAME></ROW></ROWSET>'
);
```

DBMS_XMLSAVE_CLOSECONTEXT

Эта процедура закрывает/освобождает выделенный контекст сохранения.

Входящие параметры:

- ctxhdl (INT4) — дескриптор контекста, который необходимо закрыть и удалить из памяти.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Преобразование XML-данных в типы баз данных.

Примеры запросов:

```
CALL dbms_xmlsave_closecontext(0);
```

5.2.30. Пакет DBMS_XMLSTORE

DBMS_XMLSTORE_NEWCONTEXT

Эта функция создает контекст сохранения и возвращает дескриптор контекста.

Входящие параметры:

- targettable (VARCHAR2) — имя целевой таблицы или представления, для которой создается контекст сохранения XML.

Выходные параметры:

- INTEGER - дескриптор контекста, который будет использоваться в последующих вызовах функций пакета.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
SELECT dbms_xmlstore_newcontext('public.employees');
```

DBMS_XMLSTORE_CLOSECONTEXT

Эта функция закрывает/освобождает выделенный контекст сохранения.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста, который необходимо закрыть и удалить из памяти.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
CALL dbms_xmlstore_closecontext(1);
```

DBMS_XMLSTORE_SETROWTAG

Эта функция присваивает имя тегу, используемому в XML-документе, для заключения XML-элементов, соответствующих записям базы данных.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.
- tag (VARCHAR2) — имя тега, ограничивающего строки в передаваемом XML-документе.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
CALL dbms_xmlstore_setrowtag(1, 'employee_row');
```

DBMS_XMLSTORE_SETKEYCOLUMN

Эта функция добавляет столбец в список ключевых столбцов.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.
- colname (VARCHAR2) — имя столбца, который будет использоваться в качестве ключевого для идентификации строк при операциях обновления или удаления.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
CALL dbms_xmlstore_setkeycolumn(1, 'emp_id');
```

DBMS_XMLSTORE_CLEARKEYCOLUMNLIST

Эта функция очищает список ключевых столбцов.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
CALL dbms_xmlstore_clearkeycolumnlist(1);
```

DBMS_XMLSTORE_SETUPDATECOLUMN

Эта функция добавляет столбец в список столбцов для обновления.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.
- colname (VARCHAR2) — имя столбца, который должен быть включен в список обновляемых или вставляемых столбцов.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
CALL dbms_xmlstore_setupdatecolumn(1, 'salary');
```

DBMS_XMLSTORE_CLEARUPDATECOLUMNLIST

Эта функция очищает список столбцов для обновления.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
CALL dbms_xmlstore_clearupdatecolumnlist(1);
```

DBMS_XMLSTORE_INSERTXML

Эта функция вставляет XML-документ в таблицу, указанную при создании контекста, и возвращает количество вставленных строк.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.
- xmldoc (VARCHAR2 или XMLTYPE) — XML-документ, содержащий данные для вставки в таблицу.

Выходные параметры:

- INTEGER - количество успешно вставленных строк.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
SELECT dbms_xmlstore_insertxml(1,
'<ROWSET><ROW><EMP_ID>100</EMP_ID><NAME>Ivan</NAME></ROW></ROWSET>');
```

DBMS_XMLSTORE_UPDATEXML

Эта функция обновляет указанную при создании контекста таблицу данными из XML-документа и возвращает количество обновленных строк.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.
- xmldoc (VARCHAR2 или XMLTYPE) — XML-документ, содержащий данные для обновления таблицы.

Выходные параметры:

- INTEGER - количество успешно обновленных строк.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
SELECT dbms_xmlstore_updatexml(1,
'<ROWSET><ROW><EMP_ID>100</EMP_ID><SALARY>5000</SALARY></ROW></ROWSET>');
```

DBMS_XMLSTORE_DELETEXML

Эта функция удаляет записи, указанные в данных XML-документа, из таблицы, указанной при создании контекста, и возвращает количество удаленных строк.

Входящие параметры:

- ctxhdl (INTEGER) — дескриптор контекста.
- xmldoc (VARCHAR2 или XMLTYPE) — XML-документ, содержащий строки для удаления из таблицы.

Выходные параметры:

- INTEGER - количество успешно удаленных строк.

Типичные сценарии использования:

- Сохранение XML-данных в реляционных таблицах.

Примеры запросов:

```
SELECT dbms_xmlstore_deletexml(1, '<ROWSET><ROW><EMP_ID>100</EMP_ID></ROW></ROWSET>');
```

5.2.31. Пакет OWA_MATCH

OWA_MATCH_MATCH

Эта функция используется для проверки строк или URL-адресов на соответствие спискам исключения/включения.

Входящие параметры:

- p_string (VARCHAR2) - Строка, в которой выполняется поиск совпадений.
- p_simple_pattern (VARCHAR2 DEFAULT NULL) - Массив строк, представляющих

простые шаблоны. Поиск совпадений выполняется без учета регистра (аналогично оператору ILIKE).

- p_complex_pattern (VARCHAR2 DEFAULT NULL): Массив строк, представляющих сложные регулярные выражения. Поиск выполняется без учета регистра (внутренне используется флаг i).
- p_use_special_chars (BOOLEAN DEFAULT TRUE): Флаг, указывающий на необходимость проверки наличия в строке специальных символов. Если установлен в TRUE, функция завершит поиск и вернет TRUE при наличии в p_string хотя бы одного из следующих символов: пробел, `@`, `*`, `(`, `)`, `+`, `-`, `/`, `=`, `\`, `<`, `>`, `;`, `:`, `"`, `|`, `&`, `?`, `{`, `}`, `[`, `]`, `'`, табуляция (`\t`), перевод строки (`\n`), возврат каретки (`\r`), перевод формата (`\f`).

Выходные параметры:

- BOOLEAN - TRUE, если в строке найден хотя бы один из указанных специальных символов (при p_use_special_chars = TRUE), либо если найдено совпадение хотя бы с одним из простых или сложных шаблонов. В противном случае возвращает FALSE. Если значение p_string равно NULL, функция возвращает NULL.

Типичные сценарии использования:

- Проверка строк или URL-адресов на соответствие спискам исключения/включения.

Примеры запросов:

```
-- Поиск с использованием простых шаблонов
SELECT OWA_MATCH_MATCH('hello world', ARRAY['%world%'], NULL, FALSE);

-- Поиск с использованием сложных регулярных выражений
SELECT OWA_MATCH_MATCH('test@example.com', NULL, ARRAY['^[a-z0-9._%+-]+@[a-z0-9.-
]+\.[a-z]{2,}$'], FALSE);

-- Проверка на наличие специальных символов (по умолчанию p_use_special_chars = TRUE)
SELECT OWA_MATCH_MATCH('password123!', NULL, NULL);
```

5.2.32. Пакет OWA_PATTERN

OWA_PATTERN_MATCH

Эта функция определяет, содержит ли строка указанный шаблон. Шаблон может содержать регулярные выражения. Если несколько перекрывающихся строк соответствуют регулярному выражению, функция выбирает наиболее длинное совпадение.

Входящие параметры:

- line (VARCHAR2) - Строка, в которой выполняется поиск.
- pat (VARCHAR2 или INT4) - Регулярное выражение или идентификатор предварительно скомпилированного шаблона.
- flags (VARCHAR2 DEFAULT NULL) - Строка, содержащая модификаторы регулярного выражения. Может включать один или несколько символов:
  - i — поиск без учета регистра (ignore case).

Выходные параметры:

- BOOLEAN - TRUE, если шаблон найден в строке, иначе FALSE.

Типичные сценарии использования:

- Поиск текстовых шаблонов внутри строк и замена найденных строк другими строками.

Примеры запросов:

```
SELECT OWA_PATTERN_MATCH('Oracle and PostgreSQL', 'postgre', 'i');
```

OWA_PATTERN_AMATCH

Эта функция определяет, встречается ли шаблон в определенном месте строки.

Входящие параметры:

- line (VARCHAR2) - Строка, в которой выполняется поиск.
- from_loc (INTEGER) - Позиция символа в строке line, с которой должен начаться поиск (начиная с 1).
- pat (VARCHAR2 или INT4) - Регулярное выражение или идентификатор предварительно скомпилированного шаблона.
- flags (VARCHAR2 DEFAULT NULL) - Строка, содержащая модификаторы

регулярного выражения. Может включать один или несколько символов:

- i — поиск без учета регистра (ignore case).

Выходные параметры:

- INTEGER - индекс символа, следующего за концом найденного совпадения. Если совпадение не найдено, возвращает 0.

Типичные сценарии использования:

- Поиск текстовых шаблонов внутри строк и замена найденных строк другими строками.

Примеры запросов:

```
SELECT OWA_PATTERN_AMATCH('123abc456', 1, 'abc', 'i');
```

OWA_PATTERN_CHANGE

Эта функция выполняет поиск и замену строк или данных типа multi_line. Если несколько перекрывающихся строк соответствуют регулярному выражению, эта подпрограмма выбирает наиболее длинное совпадение.

Входящие параметры:

- line (VARCHAR2) - Исходная строка, переданная для изменения.
- from_str (VARCHAR2 или INT4) - Регулярное выражение (шаблон поиска) или идентификатор предварительно скомпилированного шаблона.
- to_str (VARCHAR2) - Строка, на которую будут заменены найденные совпадения.
- flags (VARCHAR2 DEFAULT NULL) - Строка, содержащая модификаторы регулярного выражения. Может включать один или несколько символов:
  - i — поиск без учета регистра (ignore case).
  - g — глобальная замена (global). Если флаг указан, заменяются все найденные вхождения шаблона from_str в строке. Если флаг отсутствует, заменяется только первое найденное вхождение.

Выходные параметры:

- line (VARCHAR2): Измененная строка с выполненными заменами.
- INTEGER - Количество произведенных замен.

Типичные сценарии использования:

- Поиск текстовых шаблонов внутри строк и замена найденных строк другими строками.

Примеры запросов:

```
DO $$
DECLARE
    v_text VARCHAR2 := 'Oracle to PostgreSQL migration';
    v_changes INTEGER;
BEGIN
    v_changes := OWA_PATTERN_CHANGE(v_text, 'Oracle', 'Postgres', 'g');
    RAISE NOTICE 'Изменено вхождений: %, Результат: %', v_changes, v_text;
END $$;
```

OWA_PATTERN_GETPAT

Эта функция преобразует строку типа VARCHAR2 в тип данных OWA_PATTERN.PATTERN

Входящие параметры:

- arg (VARCHAR2) - Строка, содержащая регулярное выражение для компиляции и сохранения.

Выходные параметры:

- INT4 - Уникальный идентификатор (токен) скомпилированного шаблона, который затем можно передавать в другие функции пакета вместо строкового паттерна.

Типичные сценарии использования:

- Поиск текстовых шаблонов внутри строк и замена найденных строк другими строками.

Примеры запросов:

```
DO $$
DECLARE
    v_pattern_id INT4;
    v_is_match BOOLEAN;
BEGIN
    OWA_PATTERN_GETPAT('^[a-zA-Z]+$', v_pattern_id);
    v_is_match := OWA_PATTERN_MATCH('PostgreSQL', v_pattern_id);
    RAISE NOTICE 'Соответствие шаблону: %', v_is_match;
END $$;
```

5.2.33. Пакет OWA_TEXT

Пакет OWA_TEXT содержит подпрограммы для работы со строками и текстом. В частности, он предоставляет составные типы данных и методы для манипуляции большими фрагментами текста, их построения и вывода.

Типы данных

MULTI_LINE

Тип данных MULTI_LINE используется для хранения больших объемов текста в виде массива строк.

```
CREATE TYPE multi_line AS (
    rows        VARCHAR2[],
    num_rows    INTEGER,
    partial_row BOOLEAN
);
```

ROW_LIST

Тип данных ROW_LIST предназначен для хранения массивов числовых значений (идентификаторов строк).

```
CREATE TYPE row_list AS (
    rows     INTEGER[],
    num_rows INTEGER
);
```

OWA_TEXT_ADD2MULTI

Эта функция добавляет текст к существующей переменной типа MULTI_LINE.

Входящие параметры:

- stream (VARCHAR2): Текст, который необходимо добавить.
- mline (MULTI_LINE): Переменная типа MULTI_LINE. На выходе будет содержать объединенный текст.
- continue (BOOLEAN DEFAULT FALSE): Флаг, используемый во второй перегрузке. Если установлен в TRUE, подпрограмма добавит текст в конец последней строки (при условии, что ее размер не превышает лимит в 32767 байт). Если FALSE, текст будет помещен в новую строку.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Работа с большими объёмами текста.

Примеры запросов:

```
DO LANGUAGE plpgsql $$
DECLARE
    mline RECORD;
BEGIN
    mline := OWA_TEXT_NEW_MULTI();
    PERFORM OWA_TEXT_ADD2MULTI('AAAAA', mline);
END;
$$;
```

OWA_TEXT_NEW_MULTI

Создает новую пустую переменную типа MULTI_LINE. Может вызываться как функция или как процедура.

Входящие параметры:

- mline (MULTI_LINE): Выходной параметр, содержащий новую пустую структуру MULTI_LINE (в случае вызова в виде процедуры).

Выходные параметры:

- Возвращает пустой MULTI_LINE.

Типичные сценарии использования:

- Работа с большими объёмами текста.

Примеры запросов:

```
DO LANGUAGE plpgsql $$
DECLARE
    mline RECORD;
BEGIN
    mline := OWA_TEXT_NEW_MULTI();
    PERFORM OWA_TEXT_ADD2MULTI('AAAAA', mline);
END;
$$;
```

OWA_TEXT_NEW_ROW_LIST

Создает новую пустую переменную типа `ROW_LIST`. Может вызываться как функция или как процедура.

Входящие параметры:

- rlist (MULTI_LINE) - Выходной параметр, содержащий новую пустую структуру `ROW_LIST` (в случае вызова в виде процедуры).

Выходные параметры:

- Возвращает пустой ROW_LIST.

Типичные сценарии использования:

- Работа с большими объёмами текста.

Примеры запросов:

```
FUNCTION owa_text_new_row_list RETURN row_list;

PROCEDURE owa_text_new_row_list(
    rlist OUT row_list
);
```

OWA_TEXT_PRINT_MULTI

Печатает содержимое поля `rows` из переменной `MULTI_LINE`, используя процедуры пакета HTP (в частности HTP_PRINT).

Входящие параметры:

- mline (MULTI_LINE): Структура MULTI_LINE, содержимое которой требуется вывести на печать.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Работа с большими объёмами текста.

OWA_TEXT_PRINT_ROW_LIST

Печатает содержимое поля rows из переменной ROW_LIST, используя процедуры пакета HTP.

Входящие параметры:

- rlist (ROW_LIST): Структура ROW_LIST, содержимое которой требуется вывести на печать.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Работа с большими объёмами текста.

OWA_TEXT_STREAM2MULTI

Преобразует переданную строку (stream) в структуру MULTI_LINE.

Входящие параметры:

Входные параметры: * stream (VARCHAR2) - Исходная текстовая строка для конвертации. * mline (MULTI_LINE) - Возвращаемая структура MULTI_LINE, содержащая разбитый текст. * len (INTEGER): Лимит длины символов исходной

строки, которые должны быть обработаны. Усекает строку stream до len байт перед её добавлением.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Работа с большими объёмами текста.

5.2.34. Пакет OWA_UTIL

OWA_UTIL_MIME_HEADER

Эта функция позволяет изменить MIME-заголовок, который возвращает скрипт по умолчанию (обычно 'text/html').

Входящие параметры:

- ccontent_type (VARCHAR2) - MIME-тип.
- bclose_header (BOOLEAN) - Если `TRUE`, отправляется один символ новой строки для закрытия HTTP-заголовка.
- ccharset (VARCHAR2) - Кодировка символов (добавляется к заголовку Content-Type только для текстовых типов).

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
CALL owa_util_mime_header('text/plain', FALSE, 'UTF-8');
```

OWA_UTIL_REDIRECT_URL

Эта функция предписывает серверу приложений выполнить перенаправление

на указанный URL-адрес.

Входящие параметры:

- curl (VARCHAR2) - URL для перехода.
- bclose_header (BOOLEAN) - Если TRUE, отправляется один символ новой строки для закрытия HTTP-заголовка.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
CALL owa_util_redirect_url('https://example.com');
```

OWA_UTIL_SHOWSOURCE

Эта функция печатает исходный код указанной процедуры, функции или пакета.

Входящие параметры:

- cname (VARCHAR2) - Имя функции, процедуры или пакета для вывода исходного кода.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
CALL owa_util_showsource('my_procedure');
```

OWA_UTIL_SIGNATURE

Эта функция генерирует HTML-код, отображающий информацию о подписи PL/SQL, на основе которой была создана текущая страница.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
CALL owa_util_signature();
```

OWA_UTIL_STATUS_LINE

Эта процедура отправляет стандартный код состояния HTTP клиенту.

Входящие параметры:

- nstatus (NUMBER) - HTTP статус код.
- creason (VARCHAR2) - Строка с причиной для статус кода (если NULL, строка не добавляетсяowa_util_status_line).
- bclose_header (BOOLEAN) - Если TRUE, отправляется один символ новой строки для закрытия HTTP-заголовка.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
CALL owa_util_status_line(404, 'Not Found', TRUE);
```

OWA_UTIL_TABLEPRINT

Эта функция генерирует HTML-таблицу (или предварительно отформатированный текст) из указанной таблицы базы данных.

Входящие параметры:

- ctable (VARCHAR2) - Имя таблицы.
- cattributes (VARCHAR2) - Атрибуты HTML таблицы.
- ntable_type (NUMBER) - Тип таблицы (например, 1 для HTML).
- ccolumns (VARCHAR2) - Колонки для выборки (по умолчанию '*').
- cclauses (VARCHAR2) - Дополнительные предложения SQL (например, WHERE или ORDER BY).
- ccol_aliases (VARCHAR2) - Псевдонимы колонок.
- nrow_min (NUMBER) - Первая возвращаемая строка (начиная с 0).
- nrow_max (NUMBER) - Последняя возвращаемая строка.

Выходные параметры:

- BOOLEAN - TRUE в случае успеха.

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
SELECT owa_util_tableprint('employees');
```

OWA_UTIL_TODATE

Эта функция преобразует массив строк (тип DATETYPE) в стандартное значение типа DATE.

Входящие параметры:

- p_datearray (VARCHAR2) - Массив VARCHAR2 (DATETYPE), представляющий дату.

Выходные параметры:

- DATE - Полученная дата.

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
SELECT owa_util_todate(ARRAY['2023', '05', '10', '14', '30', '00']::varchar2[]);
```

OWA_UTIL_HTTP_HEADER_CLOSE

Отправляет символ новой строки для закрытия HTTP-заголовка.

Входящие параметры:

- Отсутствуют.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Получение значений переменных среды CGI, вывод данных, возвращаемых клиенту, и вывод результатов запроса в HTML-таблице.

Примеры запросов:

```
PERFORM owa_util_http_header_close();
```

5.2.35. Пакет DBMS_HTP

HTP_ADDRESS

Эта функция генерирует теги <ADDRESS> и </ADDRESS> в HTML-буффере, которые указывают адрес, автора и подпись документа.

Входящие параметры:

- cvalue (VARCHAR2) – строка, которая находится между тегами <ADDRESS> и </ADDRESS>.
- cnowrap (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, атрибут NOWRAP включается в тег.
- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование пары тегов <ADDRESS> и </ADDRESS> в HTML документе.

Примеры запросов:

```
select htp_address('618 W Fairview Ave, Dayton, OH 45406, USA', 'yes', 'left',
'myAttr=value');
```

HTP_ANCHOR

Эта функция генерирует HTML-теги <A> и </A>, которые указывают источник или назначение гипертекстовой ссылки.

Входящие параметры:

- curl (VARCHAR2) – ссылка, значение атрибута HREF.
- ctext (VARCHAR2 DEFAULT NULL) – Текст между тегами <A> и </A>.
- cname (VARCHAR2 DEFAULT NULL) – Значение атрибута NAME.

- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование пары тегов <A> и </A> в HTML документе.

Примеры запросов:

```
select htp_anchor('https://some_random.site', 'some_random.site', 'ab1211',
'myAttr=value');
```

HTP_ANCHOR2

Эта функция генерирует HTML-теги <A> и </A>, которые указывают источник или назначение гипертекстовой ссылки. Отличие этой функции от функции ANCHOR заключается в том, что она включает атрибут target и, следовательно, может использоваться для фрейма.

Входящие параметры:

- curl (VARCHAR2) – ссылка, значение атрибута HREF.
- ctext (VARCHAR2 DEFAULT NULL) – Текст между тегами <A> и </A>.
- cname (VARCHAR2 DEFAULT NULL) – Значение атрибута NAME.
- ctarget (VARCHAR2 DEFAULT NULL) – Значение атрибута TARGET.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование пары тегов <A> и </A> в HTML документе.

Примеры запросов:

```
select htp_anchor2('https://some_random.site', 'some_random.site', 'ab1211',
'_parent', 'myAttr=value');
```

HTP_APPLETCLOSE

Эта функция закрывает вызов апплета тегом </APPLET>.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие апплета тегом </APPLET>.

Примеры запросов:

```
select htp_appletclose();
```

HTP_APPLETOPEN

Эта функция генерирует тег <APPLET>, который подразумевает запуск Java-апплета. Вызов апплета завершается с помощью процедуры HTP_APPLETCLOSE, которая генерирует тег </APPLET>.

Входящие параметры:

- ccode (VARCHAR2) – Значение атрибута CODE, который указывает имя класса апплета.
- cheight (NUMBER) – Значение атрибута HEIGHT.
- cwidth (NUMBER) – Значение атрибута WIDTH.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Начало формирования тега <APPLET>.

Примеры запросов:

```
select htp_applet('testclass.class', 100, 200, 'CODEBASE="/ows-applets"');
```

HTP_AREA

Эта функция генерирует тег <AREA>, который определяет область внутри карты изображения (тег <map>), в которой есть предварительно определённые области, доступные для кликов.

Входящие параметры:

- ccoords (VARCHAR2) – Значение атрибута COORDS.
- cshape (VARCHAR2 DEFAULT NULL) – Значение атрибута SHAPE.
- chref (VARCHAR2 DEFAULT NULL) – Значение атрибута HREF.
- cnohref (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, атрибут NOHREF включается в тег.
- ctarget (VARCHAR2 DEFAULT NULL) – Значение атрибута TARGET.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <AREA> в HTML документе.

Примеры запросов:

```
select htp_area('0,95,128,0,128,137', 'poly', 'https://some.where', NULL,  '_parent',
'myAttr=value');
```

HTP_BLOCKQUOTE

Эта функция генерирует тег <BLOCKQUOTE>, который отмечает начало цитируемого фрагмента текста. Конец фрагмента текста отмечается с помощью функции HTP_BLOCKQUOTECLOSE.

Входящие параметры:

- cnowrap (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, атрибут NOWRAP включается в тег.
- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <BLOCKQUOTE> в HTML документе.

Примеры запросов:

```
select htp_blockquote('yes', 'left', 'myAttr=value');
```

HTP_BLOCKQUOTECLOSE

Эта функция генерирует тег </BLOCKQUOTE>, который обозначает конец раздела цитируемого текста.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела цитируемого текста тегом </BLOCKQUOTE>.

Примеры запросов:

```
select htp_blockquoteclose();
```

HTP_BODYOPEN

Эта функция генерирует тег <BODY>, который отмечает начало раздела <body> HTML-документа. Конец раздела <body> отмечается с помощью процедуры HTP_BODYCLOSE.

Входящие параметры:

- cbackground (VARCHAR2 DEFAULT NULL) – Значение атрибута BACKGROUND, указывающее графический файл, используемый в качестве фона документа.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <BODY> в HTML документе.

Примеры запросов:

```
select htp_bodyopen('no-repeat url("/images/lizard.png")', 'myAttr=value');
```

HTP_BODYCLOSE

Эта функция генерирует тег </BODY>, который обозначает конец секции BODY.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие секции BODY тегом </BODY>.

Примеры запросов:

```
select htp_bodyclose();
```

HTP_CENTEROPEN

Эта функция генерирует тег <CENTER>, который обозначает открытие секции CENTER. Для закрытия секции используется HTP_CENTERCLOSE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Открытие секции CENTER тегом <CENTER>.

Примеры запросов:

```
select htp_centeropen();
```

HTP_CENTERCLOSE

Эта функция генерирует тег </CENTER>, который обозначает конец секции CENTER.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие секции CENTER тегом </CENTER>.

Примеры запросов:

```
select htp_centerclose();
```

HTP_DIRLISTOPEN

Эта функция генерирует тег <DIR>, которая начинает раздел списка каталогов. Раздел списка каталогов завершается процедурой HTP_DIRLISTCLOSE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Открытие раздела списка каталогов тегом <DIR>.

Примеры запросов:

```
select htp_dirlistopen();
```

HTP_DIRLISTCLOSE

Эта функция генерирует тег </DIR>, который обозначает конец раздела списка каталогов.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела списка каталогов тегом </DIR>.

Примеры запросов:

```
select htp_dirlistclose();
```

HTP_DLISTOPEN

Эта функция генерирует тег <DL>, который начинает список определений. Завершается список определений с помощью процедуры HTP_DLISTCLOSE.

Входящие параметры:

- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <DL> в HTML документе.

Примеры запросов:

```
select htp_dlistopen('left', 'myAttr=value');
```

HTP_DLISTCLOSE

Эта функция генерирует тег </DL>, который обозначает конец списка определений.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие списка определений тегом </DL>.

Примеры запросов:

```
select htp_dlistclose();
```

HTP_DLISTTERM

Эта функция генерирует тег <DT>, который определяет термин в списке определений <DL>.

Входящие параметры:

- ctext (VARCHAR2 DEFAULT NULL) – Термин.
- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <DT> в HTML документе.

Примеры запросов:

```
select htp_dlistterm('OOP', 'left', 'myAttr=value');
```

HTP_EM

Эта функция генерирует теги <EM> и </EM>, которые определяют текст, подлежащий выделению.

Входящие параметры:

- ctext (VARCHAR2) – Текст, который подлежит выделению.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тегов <EM> и </EM> в HTML документе.

Примеры запросов:

```
select htp_em('Warning!', 'myAttr=value');
```

HTP_EMPHASIS

Эта функция генерирует теги <EM> и </EM>, которые определяют текст, подлежащий выделению. Абсолютно аналогична функции HTP_EM.

Входящие параметры:

- ctext (VARCHAR2) – Текст, который подлежит выделению.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тегов <EM> и </EM> в HTML документе.

Примеры запросов:

```
select htp_emphasis('Warning!', 'myAttr=value');
```

HTP.ESCAPE_ORACLE_FULL

Функция преобразует символы <, >, & и другие в HTML-сущности для безопасного отображения.

Входящие параметры:

- CTEXT (VARCHAR2): Строка для преобразования.

Выходные параметры:

- VARCHAR2 - экранированная строка

Типичные сценарии использования:

- экранирование символов в строке

Примеры запросов:

```
select htp_escape_oracle_full('<Apple> Яблоко');
```

HTP_FONTOPEN

Эта функция генерирует тег <FONT>, который отмечает начало раздела текста с заданными характеристиками шрифта. Конец раздела текста отмечается с помощью функции HTP_FONTCLOSE.

Входящие параметры:

- ccolor (VARCHAR2 DEFAULT NULL) – Значение атрибута COLOR.
- cface (VARCHAR2 DEFAULT NULL) – Значение атрибута FACE.
- csize (VARCHAR2 DEFAULT NULL) – Значение атрибута SIZE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <FONT> в HTML документе.

Примеры запросов:

```
select htp_fontopen('green', 'Roboto Condensed, Sans Serif', '6', 'myAttr=value');
```

HTP_FONTCLOSE

Эта функция генерирует тег </FONT>, который отмечает конец фрагмента текста

с заданными характеристиками шрифта.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие фрагмента текста с заданными характеристиками шрифта тегом </FONT>.

Примеры запросов:

```
select htp_fontclose();
```

HTP_FORMCHECKBOX

Эта функция генерирует тег <INPUT> с типом TYPE="checkbox", который вставляет элемент флажка в форму. Элемент флажка — это кнопка, которую пользователь включает или выключает.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- cvalue (VARCHAR2 DEFAULT 'ON') – Значение атрибута VALUE.
- cchecked (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, к тегу добавляется атрибут CHECKED.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с атрибутом TYPE="checkbox" в HTML документе.

Примеры запросов:

```
select htp_formcheckbox('chk32', '1', 'yes', 'myAttr=value');
```

HTP_FORMFILE

Эта функция генерирует тег <INPUT> с типом TYPE="file", который вставляет элемент формы для загрузки файла. Он используется для загрузки файлов на заданную страницу.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- caccept (VARCHAR2 DEFAULT NULL) – Список MIME-типов для загрузки, разделённый запятыми.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с атрибутом TYPE="file" в HTML документе.

Примеры запросов:

```
select htp_formfile('get_image_2', 'image/png, image/jpeg', 'myAttr=value');
```

HTP_FORMHIDDEN

Эта функция генерирует тег <INPUT> с параметром TYPE="hidden", который вставляет скрытый элемент формы. Этот элемент не виден пользователю. Он отправляет дополнительные значения в скрипт.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- cvalue (VARCHAR2 DEFAULT NULL) – Значение атрибута VALUE.

- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с атрибутом TYPE="hidden" в HTML документе.

Примеры запросов:

```
select htp_formhidden('hid_data434', 'grab_cookie', 'myAttr=value');
```

HTP_FORMIMAGE

Эта функция генерирует тег <INPUT> с типом TYPE="image", который создает поле для изображения, по которому пользователь щелкает, чтобы немедленно отправить форму. Координаты выбранной точки измеряются в пикселях и возвращаются (вместе с остальным содержимым формы) в виде двух пар имя/значение. Координата x отправляется под именем поля с добавлением .x, а координата y — с добавлением .y. Любой атрибут VALUE игнорируется.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- csrc (VARCHAR2) – Значение атрибута SRC, указывающее на файл изображения.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с атрибутом TYPE="image" в HTML документе.

Примеры запросов:

```
select htp_formimage('img_04', '/images/login-button.png', 'left', 'myAttr=value');
```

HTP_FORMOPEN

Эта функция генерирует тег <FORM>, который отмечает начало раздела формы в HTML-документе. Конец раздела формы отмечается с помощью процедуры HTP_FORMCLOSE.

Входящие параметры:

- curl (VARCHAR2) – URL-адрес скрипта WRB или CGI, куда отправляется содержимое формы. Этот параметр обязателен.
- cmethod (VARCHAR2 DEFAULT POST) – Значение атрибута METHOD. Значение может быть "GET" или "POST".
- ctarget (VARCHAR2 DEFAULT NULL) – Значение атрибута TARGET.
- cenctype (VARCHAR2 DEFAULT NULL) – Значение атрибута ENCTYPE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <FORM> в HTML документе.

Примеры запросов:

```
select htp_formopen('', 'POST', '_blank', 'text/plain', 'myAttr=value');
```

HTP_FORMCLOSE

Эта функция генерирует тег </FORM>, который отмечает конец раздела формы в HTML-документе.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела формы тегом </FORM>.

Примеры запросов:

```
select htp_formclose();
```

HTP_FORMPASSWORD

Эта функция генерирует тег <INPUT> с типом TYPE="password", который создает текстовое поле для ввода в одну строку. Когда пользователь вводит текст в это поле, каждый символ обозначается звездочкой. Это используется для ввода паролей.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- csize (VARCHAR2) – Значение атрибута SIZE.
- cmaxlength (VARCHAR2 DEFAULT NULL) – Значение атрибута MAXLENGTH.
- cvalue (VARCHAR2 DEFAULT NULL) – Значение атрибута VALUE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с атрибутом TYPE="password" в HTML документе.

Примеры запросов:

```
select htp_formpassword('pass123', '20', '20', '', 'myAttr=value');
```

HTP_FORMRADIO

Эта функция генерирует тег <INPUT> с типом TYPE="radio", который создает переключатель на HTML-форме. В наборе переключателей пользователь выбирает только один. Каждый переключатель в одном наборе имеет одно и то же имя, но разные значения. Выбранный переключатель генерирует пару имя/значение.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- cvalue (VARCHAR2) – Значение атрибута VALUE.
- cchecked (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не NULL, к тегу добавляется атрибут CHECKED.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с атрибутом TYPE="radio" в HTML документе.

Примеры запросов:

```
select htp_formradio('radio_44', 'email', 'yes', 'myAttr=value');
```

HTP_FORMRESET

Эта функция генерирует тег <INPUT> с параметром TYPE="reset", который создает кнопку, при нажатии на которую поля формы сбрасываются до их исходных значений.

Входящие параметры:

- cvalue (VARCHAR2 DEFAULT 'Reset') – Значение атрибута VALUE.

- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с атрибутом TYPE="reset" в HTML документе.

Примеры запросов:

```
select htp_formreset('reset the form', 'myAttr=value');
```

HTP_FORMSELECTOPEN

Эта функция генерирует тег <SELECT>, который создаёт элемент формы «Select». Элемент формы «Select» представляет собой список, в котором пользователь выбирает одно или несколько значений. Конец элемента формы «Select» обозначается с помощью процедуры HTP_FORMSELECTCLOSE. Значения вставляются с помощью процедуры HTP_FORMSELECTOPTION.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- cprompt (VARCHAR2 DEFAULT NULL) – Строка, предшествующая полю списка.
- nsize (NUMBER DEFAULT NULL) – Значение атрибута SIZE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <SELECT> в HTML документе.

Примеры запросов:

```
select htp_formselectopen('select', 'select section', '5', 'myAttr=value');
```

HTP_FORMSELECTCLOSE

Эта функция генерирует тег </SELECT>, который отмечает конец элемента формы Select.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие элемента формы Select тегом </SELECT>.

Примеры запросов:

```
select htp_formselectclose();
```

HTP_FORMSELECTOPTION

Эта функция генерирует тег <OPTION>, который представляет один из вариантов выбора в элементе <SELECT>.

Входящие параметры:

- cvalue (VARCHAR2) – Текстовое поле опции.
- cselected (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, к тегу добавляется атрибут SELECTED.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <OPTION> внутри тега <SELECT> в HTML документе.

Примеры запросов:

```
select htp_formselectoption('select this one', 'yes', 'myAttr=value');
```

HTP_FORMSUBMIT

Эта функция генерирует тег <INPUT> с параметром TYPE="submit", который создает кнопку, при нажатии на которую отправляется форма. Если у кнопки есть атрибут NAME, она добавляет пару "имя/значение" к отправляемым данным.

Входящие параметры:

- cname (VARCHAR2 DEFAULT NULL) – Значение атрибута NAME.
- cvalue (VARCHAR2 DEFAULT 'SUBMIT') – Значение атрибута VALUE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с параметром TYPE="submit" в HTML документе.

Примеры запросов:

```
select htp_formsubmit('button_13', 'Send Request', 'myAttr=value');
```

HTP_FORMTEXT

Эта функция генерирует тег <INPUT> с типом TYPE="text", который создает поле для одной строки текста.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- csize (VARCHAR2 DEFAULT NULL) – Значение атрибута SIZE.

- cmaxlength (VARCHAR2 DEFAULT NULL) – Значение атрибута MAXLENGTH.
- cvalue (VARCHAR2 DEFAULT NULL) – Значение атрибута VALUE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <INPUT> с параметром TYPE="text" в HTML документе.

Примеры запросов:

```
select htp_formtext('txt_label', '18', '20', 'empty label', 'myAttr=value');
```

HTP_FORMTEXTAREA

Эта функция генерирует тег <TEXTAREA>, который создает текстовое поле, не содержащее предопределенного текста. Это поле позволяет вводить несколько строк текста.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htp_formtextarea('txt_area_2', 3, 3, 'center', 'myAttr=value');
```

HTP_FORMTEXTAREA2

Эта функция генерирует тег <TEXTAREA>, который создает текстовое поле, не содержащее предопределенного текста. Это поле позволяет вводить несколько строк текста. В отличие от функции HTP_FORMTEXTAREA, имеет параметр cwrap, позволяющий указать стиль переноса текста.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cwrap (VARCHAR2 DEFAULT NULL) – Значение атрибута WRAP.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htp_formtextarea2('txt_area_2', 3, 3, 'center', 'hard', 'myAttr=value');
```

HTP_FORMTEXTAREAOPEN

Эта функция генерирует тег <TEXTAREA>, который отмечает начало текстового поля формы. Конец текстового поля формы отмечается с помощью процедуры HTP_FORMTEXTAREACLOSE.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htp_formtextareaopen('txt_area_3', 2, 2, 'center', 'myAttr=value');
```

HTP_FORMTEXTAREAOPEN2

Эта функция генерирует тег <TEXTAREA>, который отмечает начало текстового поля формы. В отличие от функции HTP_FORMTEXTAREAOPEN, имеет параметр cwrap, позволяющий указать стиль переноса текста. Конец текстового поля формы отмечается с помощью процедуры HTP_FORMTEXTAREACLOSE.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cwrap (VARCHAR2 DEFAULT NULL) – Значение атрибута WRAP.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htp_formtextareaopen2('txt_area_3', 2, 2, 'center', 'hard', 'myAttr=value');
```

HTP_FORMTEXTAREACLOSE

Эта функция генерирует тег </TEXTAREA>, который завершает элемент текстовой формы.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие текстовой формы Select тегом </TEXTAREA>.

Примеры запросов:

```
select htp_formtextareaclose();
```

HTP_FRAME

Эта функция генерирует тег <FRAME>, который начинает описание характеристик фрейма, созданного тегом <FRAMESET>.

Входящие параметры:

- csrc (VARCHAR2) – URL, который будет отображён в фрейме.
- cname (VARCHAR2) – Значение атрибута NAME.

- cmarginwidth (VARCHAR2 DEFAULT NULL) – Значение атрибута MARGINWIDTH.
- cmarginheight (VARCHAR2 DEFAULT NULL) – Значение атрибута MARGINHEIGHT.
- cscrolling (VARCHAR2 DEFAULT NULL) – Значение атрибута SCROLLING.
- cnoresize (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, к тегу добавляется атрибут NORESIZE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <FRAME> в HTML документе.

Примеры запросов:

```
select htp_frame('https://some.site', 'art site', '100', '100', 'yes', 'yes',
'myAttr=value');
```

HTP_FRAMESETOPEN

Эта функция генерирует тег <FRAMESET>, определяющий раздел набора фреймов. Конец раздела набора фреймов отмечается с помощью процедуры HTP_FRAMESETCLOSE.

Входящие параметры:

- crows (VARCHAR2 DEFAULT NULL) – Значение атрибута ROWS.
- ccols (VARCHAR2 DEFAULT NULL) – Значение атрибута COLS.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <FRAMESET> в HTML документе.

Примеры запросов:

```
select htp_framesetopen('2', '3', 'myAttr=value');
```

HTP_FRAMESETCLOSE

Эта функция генерирует тег </FRAMESET>, который завершает раздел фреймсета.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела фреймсета тегом </FRAMESET>.

Примеры запросов:

```
select htp_framesetclose();
```

HTP_HEADOPEN

Эта функция генерирует тег <HEAD>, который отмечает начало раздела заголовка HTML-документа. Конец раздела заголовка HTML-документа отмечается с помощью процедуры HTP_HEADCLOSE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Открытие раздела заголовка тегом <HEAD>.

Примеры запросов:

```
select htp_headopen();
```

HTP_HEADCLOSE

Эта функция генерирует тег </HEAD>, который обозначает конец раздела заголовка HTML-документа.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела заголовка тегом </HEAD>.

Примеры запросов:

```
select htp_headclose();
```

HTP_HEADER

Эта функция генерирует открывающие теги заголовка (от <H1> до <H6>) и соответствующие им закрывающие теги (от </H1> до </H6>).

Входящие параметры:

- nsize (NUMBER) – Уровень заголовка. Это целое число от 1 до 6.
- cheader (VARCHAR2) – Текст заголовка.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cnowrap (VARCHAR2 DEFAULT NULL) – Значение атрибута NOWRAP.
- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут

включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тегов заголовков (от <H1> до <H6>) в HTML документе.

Примеры запросов:

```
select htp_header(2, 'Great Title', 'center', 'yes', 'right', 'myAttr=value');
```

HTP_HTMLOPEN

Эта функция генерирует тег <HTML>, который отмечает начало HTML-документа. Для обозначения конца HTML-документа используется процедура HTP_HTMLCLOSE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Открытие HTML-документа тегом <HTML>.

Примеры запросов:

```
select htp_htmlopen();
```

HTP_HTMLCLOSE

Эта функция генерирует тег </HTML>, который обозначает конец HTML-документа.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие HTML-документа тегом </HTML>.

Примеры запросов:

```
select htp_htmlclose();
```

HTP_LISTINGOPEN

Эта функция генерирует тег <LISTING>, который отмечает начало раздела текста фиксированной ширины в теле HTML-страницы. Чтобы отметить конец раздела текста фиксированной ширины в теле HTML-страницы, используйте процедуру HTP_LISTINGCLOSE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Открытие раздела текста фиксированной ширины тегом <LISTING>.

Примеры запросов:

```
select htp_listingopen();
```

HTP_LISTINGCLOSE

Эта функция генерирует тег </LISTING>, который обозначает конец раздела текста фиксированной ширины HTML-документа.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела текста фиксированной ширины тегом </LISTING>.

Примеры запросов:

```
select htp_listingclose();
```

HTP_MAPOPEN

Эта функция генерирует тег <MAP>, который отмечает начало набора регионов на карте изображений на стороне клиента. Чтобы отметить конец набора регионов на карте изображений на стороне клиента, используйте процедуру HTP_MAPCLOSE.

Входящие параметры:

- cname (VARCHAR2 DEFAULT NULL) – Значение атрибута NAME.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <MAP> в HTML документе.

Примеры запросов:

```
select htp_mapopen('infographic', 'myAttr=value');
```

HTP_MAPCLOSE

Эта функция генерирует тег </MAP>, который обозначает конец секции набора

регионов на карте изображений на стороне клиента.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие секции набора регионов на карте изображений тегом </MAP>.

Примеры запросов:

```
select htp_mapclose();
```

HTP_MENULISTOPEN

Эта функция генерирует тег <MENU>, который начинает список, отображающий по одной строке каждый элемент. Чтобы завершить список такого типа, используйте процедуру HTP_MENULISTCLOSE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Открытие раздела MENU тегом <MENU>.

Примеры запросов:

```
select htp_menulistopen();
```

HTP_MENULISTCLOSE

Эта функция генерирует тег </MENU>, который обозначает конец списка,

отображающего по одной строке каждый элемент.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела MENU тегом </MENU>.

Примеры запросов:

```
select htp_menulistclose();
```

HTP_NOFRAMESOPEN

Эта функция генерирует тег <NOFRAMES>, который отмечает начало раздела без фреймов. Чтобы отметить конец раздела без фреймов, используйте процедуру HTP_FRAMESETCLOSE.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Открытие раздела без фреймов тегом <NOFRAMES>.

Примеры запросов:

```
select htp_noframesopen();
```

HTP_NOFRAMESCLOSE

Эта функция генерирует тег </NOFRAMES>, который обозначает конец раздела

без фреймов.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела без фреймов тегом </NOFRAMES>.

Примеры запросов:

```
select htp_noframesclose();
```

HTP_OLISTOPEN

Эта функция генерирует тег <OL>, который отмечает начало упорядоченного списка. Упорядоченный список представляет собой перечень пронумерованных элементов. Чтобы отметить конец списка такого типа, используйте процедуру HTP_OLISTCLOSE. Пронумерованные элементы добавляются с помощью процедуры HTP_LISTITEM.

Входящие параметры:

- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cwrap (VARCHAR2 DEFAULT NULL) – Значение атрибута WRAP.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <OL> в HTML документе.

Примеры запросов:

```
select htp_olistopen('right', 'hard', 'myAttr=value');
```

HTP_OLISTCLOSE

Эта функция генерирует тег </OL>, который обозначает конец упорядоченного списка.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие упорядоченного списка тегом </OL>.

Примеры запросов:

```
select htp_olistclose();
```

HTP_PREOPEN

Эта функция генерирует тег <PRE>, который отмечает начало раздела предварительно отформатированного текста в теле HTML-страницы. Чтобы отметить конец раздела предварительно отформатированного текста в теле HTML-страницы, используйте процедуру HTP_PRECLOSE.

Входящие параметры:

- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cwidth (VARCHAR2 DEFAULT NULL) – Значение атрибута WIDTH.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <PRE> в HTML документе.

Примеры запросов:

```
select htp_preopen('right', '100', 'myAttr=value');
```

HTP_PRECLOSE

Эта функция генерирует тег </PRE>, который обозначает конец раздела предварительно отформатированного текста.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие раздела предварительно отформатированного текста тегом </PRE>.

Примеры запросов:

```
select htp_preclose();
```

HTP_PRINT

Эти перегруженные функции генерируют указанный параметр в виде строки, завершающейся символом новой строки \n.

Входящий параметр:

- cbuf (VARCHAR2) или dbuf (DATE) или nbuf (NUMBER) – Строка, которую мы хотим добавить в буфер с символом новой строки в конце.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Добавление строки с символом новой строки в конце в HTML документ.

Примеры запросов:

```
select htp_print('My new line!');
```

HTP_PRN

Эти перегруженные функции генерируют указанный параметр в виде строки. В отличие от HTP_PRINT эти строки не завершаются символом новой строки \n.

Входящий параметр:

- cbuf (VARCHAR2) или dbuf (DATE) или nbuf (NUMBER) – Строка, которую мы хотим добавить в буфер.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Добавление строки в HTML документ.

Примеры запросов:

```
select htp_prn('My new line!');
```

HTP_TABLEOPEN

Эта функция генерирует тег <TABLE>, который обозначает начало HTML-таблицы. Чтобы определить конец HTML-таблицы, используйте процедуру HTP_TABLECLOSE.

Входящие параметры:

- cborder (VARCHAR2 DEFAULT NULL) – Значение атрибута BORDER.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cnowrap (VARCHAR2 DEFAULT NULL) – Значение атрибута NOWRAP.

- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <TABLE> в HTML документе.

Примеры запросов:

```
select htp_tableopen('2', 'left', 'yes', 'left', 'myAttr=value');
```

HTP_TABLECLOSE

Эта функция генерирует тег </TABLE>, который обозначает конец HTML-таблицы.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие HTML-таблицы тегом </TABLE>.

Примеры запросов:

```
select htp_tableclose();
```

HTP_TABLEROWOPEN

Эта функция генерирует тег <TR>, который отмечает начало новой строки в HTML-таблице. Чтобы отметить конец новой строки, используйте процедуру HTP_TABLEROWCLOSE.

Входящие параметры:

- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cvalign (VARCHAR2 DEFAULT NULL) – Значение атрибута VALIGN.
- cdp (VARCHAR2 DEFAULT NULL) – Значение атрибута DP.
- cnowrap (VARCHAR2 DEFAULT NULL) – Значение атрибута NOWRAP.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <TR> в HTML документе.

Примеры запросов:

```
select htp_tablerowopen('left', 'middle', '.', 'yes', 'myAttr=value');
```

HTP_TABLEROWCLOSE

Эта функция генерирует тег </TR>, который обозначает конец строки в HTML-таблице.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие строки в HTML-таблице тегом </TR>.

Примеры запросов:

```
select htp_tablerowclose();
```

HTP_ULISTOPEN

Эта функция генерирует тег <UL>, который отмечает начало неупорядоченного списка. Неупорядоченный список отображает элементы с помощью маркеров. Чтобы отметить конец неупорядоченного списка, используйте процедуру HTP_ULISTCLOSE. Добавляйте элементы списка с помощью процедуры HTP_LISTITEM.

Входящие параметры:

- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cwrap (VARCHAR2 DEFAULT NULL) – Значение атрибута WRAP.
- cdingbat (VARCHAR2 DEFAULT NULL) – Значение атрибута DINGBAT.
- csrc (VARCHAR2 DEFAULT NULL) – Значение атрибута SRC.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Формирование тега <UL> в HTML документе.

Примеры запросов:

```
select htp_ulistopen('left', 'yes', '*', 'image1.jpg', 'myAttr=value');
```

HTP_ULISTCLOSE

Эта функция генерирует тег </UL>, который обозначает конец неупорядоченного списка.

Входящие параметры:

- Отсутствуют

Выходные параметры:

- Отсутствуют

Типичные сценарии использования:

- Закрытие неупорядоченного списка тегом </UL>.

Примеры запросов:

```
select htp_ulistclose();
```

5.2.36. Пакет DBMS_HTF

HTF_ADDRESS

Эта функция генерирует теги <ADDRESS> и </ADDRESS> в HTML-буффере, которые указывают адрес, автора и подпись документа.

Входящие параметры:

- cvalue (VARCHAR2) – строка, которая находится между тегами <ADDRESS> и </ADDRESS>.
- cnowrap (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, атрибут NOWRAP включается в тег.
- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование пары тегов <ADDRESS> и </ADDRESS> в HTML документе.

Примеры запросов:

```
select htf_address('618 W Fairview Ave, Dayton, OH 45406, USA', 'yes', 'left',
'myAttr=value');
```

HTF_ANCHOR

Эта функция генерирует HTML-теги <A> и </A>, которые указывают источник

или назначение гипертекстовой ссылки.

Входящие параметры:

- curl (VARCHAR2) – ссылка, значение атрибута HREF.
- ctext (VARCHAR2) – Текст между тегами <A> и </A>.
- cname (VARCHAR2 DEFAULT NULL) – Значение атрибута NAME.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование пары тегов <A> и </A> в HTML документе.

Примеры запросов:

```
select htf_anchor('https://some_random.site', 'some_random.site', 'ab1211',
'myAttr=value');
```

HTF_ANCHOR2

Эта функция генерирует HTML-теги <A> и </A>, которые указывают источник или назначение гипертекстовой ссылки. Отличие этой функции от функции HTF_ANCHOR заключается в том, что она включает атрибут target и, следовательно, может использоваться для фрейма.

Входящие параметры:

- curl (VARCHAR2) – ссылка, значение атрибута HREF.
- ctext (VARCHAR2) – Текст между тегами <A> и </A>.
- cname (VARCHAR2 DEFAULT NULL) – Значение атрибута NAME.
- ctarget (VARCHAR2 DEFAULT NULL) – Значение атрибута TARGET.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование пары тегов <A> и </A> в HTML документе.

Примеры запросов:

```
select htf_anchor2('https://some_random.site', 'some_random.site', 'ab1211',
'_parent', 'myAttr=value');
```

HTF_APPLETOPEN

Эта функция генерирует тег <APPLET>, который подразумевает запуск Java-апплета. Вызов апплета завершается с помощью процедуры HTF_APPLETCLOSE, которая генерирует тег </APPLET>.

Входящие параметры:

- ccode (VARCHAR2) – Значение атрибута CODE, который указывает имя класса апплета.
- cheight (NUMBER) – Значение атрибута HEIGHT.
- cwidth (NUMBER) – Значение атрибута WIDTH.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Начало формирования тега <APPLET>.

Примеры запросов:

```
select htf_applet('testclass.class', 100, 200, 'CODEBASE="/ows-applets"');
```

HTF_AREA

Эта функция генерирует тег <AREA>, который определяет карту изображений на

стороне клиента. Тег <AREA> определяет области внутри изображения и места назначения для этих областей.

Входящие параметры:

- ccoords (VARCHAR2) – Значение атрибута COORDS.
- cshape (VARCHAR2 DEFAULT NULL) – Значение атрибута SHAPE.
- chref (VARCHAR2 DEFAULT NULL) – Значение атрибута HREF.
- cnohref (VARCHAR2 DEFAULT NULL) – Если значение этого параметра не равно NULL, атрибут NOHREF включается в тег.
- ctarget (VARCHAR2 DEFAULT NULL) – Значение атрибута TARGET.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <AREA> в HTML документе.

Примеры запросов:

```
select htf_area('0,95,128,0,128,137', 'poly', 'https://some.where', NULL,  '_parent',
'myAttr=value');
```

HTF_BODYOPEN

Эта функция генерирует тег <BODY>, который отмечает начало раздела <body> HTML-документа. Конец раздела <body> отмечается с помощью процедуры HTF_BODYCLOSE.

Входящие параметры:

- cbackground (VARCHAR2 DEFAULT NULL) – Значение атрибута BACKGROUND, указывающее графический файл, используемый в качестве фона документа.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <BODY> в HTML документе.

Примеры запросов:

```
select htf_bodyopen('no-repeat url("/images/lizard.png")', 'myAttr=value');
```

HTF_CENTER

Эта функция генерирует теги <CENTER> и </CENTER>, которые центрируют фрагмент текста на веб-странице.

Входящие параметры:

- ctext (VARCHAR2) – Текст между тегами.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Центрирование фрагмента текста на веб-странице.

Примеры запросов:

```
select htf_center('Text to be in center');
```

HTF_CITE

Эта функция генерирует теги <CITE> и </CITE>, которые указывают браузеру на необходимость отображения текста в виде цитаты.

Входящие параметры:

- ctext (VARCHAR2) – Цитируемый текст.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут

включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Цитирование текста на веб-странице.

Примеры запросов:

```
select htf_cite('Text to be citated', 'myAttr=value');
```

HTF_CODE

Эта функция генерирует теги <CODE> и </CODE>, которые указывают браузеру отображать текст моноширинным шрифтом или любым другим стилистическим способом, заданным для кода.

Входящие параметры:

- ctext (VARCHAR2) – Код, помещаемый между тегами.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Добавление кода на веб-страницу.

Примеры запросов:

```
select htf_code('print("Hello World!")', 'myAttr=value');
```

HTF_CODEPOINT

Функция преобразует символ в его Unicode кодовую точку (например, 'A' → 65).

Входящие параметры:

- P_CHAR (VARCHAR2) – символ для преобразования в код.

Выходные параметры:

- res (BINARY_INTEGER) - код символа.

Типичные сценарии использования:

- Перевод символов в числовое представление.

Примеры запросов:

```
select htf_codepoint('A');
```

HTF_COMMENT

Эта функция генерирует теги HTML-комментария <!-- -→.

Входящие параметры:

- ctext (VARCHAR2) – Текст комментария.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Добавление комментария в код веб-страницы.

Примеры запросов:

```
select htf_comment('unused element');
```

HTF_DFN

Эта функция генерирует теги <DFN> и </DFN>, которые указывают браузеру выделять текст курсивом или любым другим стилистическим способом, определённым для этого.

Входящие параметры:

- ctext (VARCHAR2) – Текст, который необходимо выделить.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Стилистическое выделение текста на веб-странице.

Примеры запросов:

```
select htf_dfn('This needs to be emphasized');
```

HTF_DIV

Эта функция генерирует тег <DIV>, который создает разделы документа.

Входящие параметры:

- calign (VARCHAR2 DEFAULT NULL) – значение атрибута ALIGN.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Создание раздела на веб-странице.

Примеры запросов:

```
select htf_div('center', 'myAttr=value');
```

HTF_DLISTDEF

Эта функция генерирует тег <DD>, который вставляет определения терминов. Используйте этот тег для списка определений <DL>. Термины помечаются тегом <DT>, а определения — тегом <DD>.

Входящие параметры:

- ctext (VARCHAR2 DEFAULT NULL) – Определение термина.
- cclear (VARCHAR2 DEFAULT NULL) – значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Заполнение списка определений на веб-странице.

Примеры запросов:

```
select htf_dlistdef('Really big animal', 'left', 'myAttr=value');
```

HTF_DLISTOPEN

Эта функция генерирует тег <DL>, который начинает список определений. Завершается список определений с помощью процедуры HTF_DLISTCLOSE.

Входящие параметры:

- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <DL> в HTML документе.

Примеры запросов:

```
select htf_dlistopen('left', 'myAttr=value');
```

HTF_DLISTTERM

Эта функция генерирует тег <DT>, который определяет термин в списке определений <DL>.

Входящие параметры:

- ctext (VARCHAR2 DEFAULT NULL) – Термин.
- cclear (VARCHAR2 DEFAULT NULL) – Значение атрибута CLEAR.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <DT> в HTML документе.

Примеры запросов:

```
select htf_dlistterm('OOP', 'left', 'myAttr=value');
```

HTF_EM

Функция генерирует теги <EM> для логического выделения текста (обычно курсив).

Входящие параметры:

- CTEXT (VARCHAR2): Текст для выделения.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
select htf_em('"qw Я"','"qw Я"'); -- <em "qw Я">&quot;qw &#x042F;&quot;</em>
```

HTF_EMPHASIS

Функция генерирует теги <EM> для логического выделения текста (синоним функции EM).

Входящие параметры:

- CTEXT (VARCHAR2): Текст для выделения.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
select htf_emphasis('"qw Я"','"qw Я"'); --  <em "qw Я">&quot;qw &#x042F;&quot;</em>
```

HTF_ESCAPE_ORACLE_FULL

Функция преобразует символы <, >, & и другие в HTML-сущности для безопасного отображения.

Входящие параметры:

- CTEXT (VARCHAR2): Строка для преобразования.

Выходные параметры:

- VARCHAR2 - экранированная строка

Типичные сценарии использования:

- экранирование символов в строке

Примеры запросов:

```
select htf_escape_oracle_full('Apple Яблоко'); -- Apple
&#x042F;&#x0431;&#x043B;&#x043E;&#x043A;&#x043E;
```

HTF_ESCAPE_SC

Эта функция преобразует символы <, >, & и другие в HTML-сущности для безопасного отображения.

Входящие параметры:

- ctext (VARCHAR2) – текст с символами для экранирования.

Выходные параметры:

- result (VARCHAR2) - строка, содержащая текст с экранированными символами.

Типичные сценарии использования:

- Необходимость экранирования символов для правильного отображения на HTML-страницах.

Примеры запросов:

```
select htf_escape_sc('<alert(1)>');
```

HTF_ESCAPE_URL

Эта функция выполняет URL-кодирование строки, заменяя специальные символы (устаревшая, рекомендуется использовать UTL_URL.ESCAPE).

Входящие параметры:

- p_url (VARCHAR2) – текст с символами для экранирования.

Выходные параметры:

- result (VARCHAR2) - строка, содержащая текст с экранированными символами.

Типичные сценарии использования:

- Необходимость экранирования символов для правильного отображения на HTML-страницах.

Примеры запросов:

```
select htf_escape_url('www.<alert(&)>.ru');
```

HTF_FONTOPEN

Функция генерирует открывающий тег <FONT> для изменения шрифта (устаревший тег, использовать CSS).

Входящие параметры:

- CCOLOR (VARCHAR2): Значение для атрибута COLOR (цвет шрифта).
- CFACE (VARCHAR2): Значение для атрибута FACE (название шрифта).
- CSIZE (VARCHAR2): Значение для атрибута SIZE (размер шрифта).
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_fontopen('red', 'Arial', '+2', 'class="highlight" style="font-
weight:bold;"');
```

HTF_FORMATCELL

Функция форматирует значения ячеек таблицы, обрабатывая числа специальным образом (выравнивание и округление).

Входящие параметры:

- COLUMNVALUE (VARCHAR2): Значение, которое нужно отформатировать в HTML-таблице.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
select htf_format_cell('"qw Я"', NULL);
```

HTF_FORMCHECKBOX

Функция создает элемент флажка (checkbox) в HTML-форме.

Входящие параметры:

- CNAME (VARCHAR2): Значение для атрибута NAME.
- CVALUE (VARCHAR2): Значение для атрибута VALUE.
- CCHECKED (VARCHAR2): Если значение этого параметра не NULL, в тег добавляется атрибут CHECKED.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formcheckbox('agree_terms', 'yes', 'CHECKED', 'class="form-check"
id="agree" required');
```

HTF_FORMFILE

Функция создает поле выбора файла для загрузки на сервер.

Входящие параметры:

- CNAME (VARCHAR2): Значение для атрибута NAME.
- CACCEPT (VARCHAR2): Список MIME-типов для загрузки через запятую.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formfile('upload', '50', 'accept=".pdf,.docx" required');
```

HTF_FORMHIDDEN

Функция создает скрытое поле формы, значение которого отправляется на сервер, но не отображается пользователю.

Входящие параметры:

- CNAME (VARCHAR2): Значение для атрибута NAME.
- CVALUE (VARCHAR2): Значение для атрибута VALUE.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formhidden('csrf_token', '7d8f9a2b3c4d5e6f', 'id="csrf"');
```

HTF_FORMIMAGE

Функция создает графическую кнопку отправки формы, при клике на которую отправляются координаты клика.

Входящие параметры:

- CNAME (VARCHAR2): Значение для атрибута NAME.
- CSRC (VARCHAR2): Значение для атрибута SRC (URL изображения).
- CALIGN (VARCHAR2): Значение для атрибута ALIGN.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formimage('submit_btn', '/images/submit.png', 'center', 'width="120"
height="40" class="btn-primary"');
```

HTF_FORMOPEN

Функция создает открывающий тег <FORM> для размещения элементов формы.

Входящие параметры:

- CURL (VARCHAR2): URL скрипта, обрабатывающего данные формы.

- CMETHOD (VARCHAR2): Значение для атрибута METHOD ("GET" или "POST").
- CTARGET (VARCHAR2): Значение для атрибута TARGET.
- CENCTYPE (VARCHAR2): Значение для атрибута ENCTYPE (тип кодирования данных).
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formopen('/submit-order', 'POST', 'target', 'multipart/form-data',
'class="order-form" id="order"');
```

HTF_FORMPASSWORD

Функция создает открывающий тег <FORM> для размещения элементов формы.

Входящие параметры:

- CNAME (VARCHAR2): Значение для атрибута NAME.
- CSIZE (VARCHAR2): Значение для атрибута SIZE (размер поля).
- CMAXLENGTH (VARCHAR2): Значение для атрибута MAXLENGTH.
- CVALUE (VARCHAR2): Значение для атрибута VALUE.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formpassword('user_pass', '20', '30', '', 'class="form-control"
id="password" placeholder="Введите пароль" required');
```

HTF_FORMRADIO

Функция создает открывающий тег <FORM> для размещения элементов формы.

Входящие параметры:

- CNAME (VARCHAR2): Значение для атрибута NAME.
- CVALUE (VARCHAR2): Значение для атрибута VALUE.
- CCHECKED (VARCHAR2): Если значение этого параметра не NULL, в тег добавляется атрибут CHECKED.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formradio('gender', 'male', 'CHECKED', 'class="form-radio" id="gender-
male"');
```

HTF_FORMRESET

Функция создает кнопку, которая сбрасывает все поля формы к исходным значениям.

Входящие параметры:

- CVALUE (VARCHAR2): Значение для атрибута VALUE (текст на кнопке).
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут

включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formreset('Очистить форму', 'class="btn-reset" id="resetBtn"
style="background:#f0f0f0;"');
```

HTF_FORMSELECTOPEN

Эта функция генерирует тег <SELECT>, который создаёт элемент формы «Select». Элемент формы «Select» представляет собой список, в котором пользователь выбирает одно или несколько значений. Конец элемента формы «Select» обозначается с помощью процедуры HTF_FORMSELECTCLOSE. Значения вставляются с помощью процедуры HTF_FORMSELECTOPTION.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- cprompt (VARCHAR2 DEFAULT NULL) – Строка, предшествующая полю списка.
- nsize (NUMBER DEFAULT NULL) – Значение атрибута SIZE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <SELECT> в HTML документе.

Примеры запросов:

```
select htf_formselectopen('select', 'select section', '5', 'myAttr=value');
```

HTF_FORMSELECTOPTION

Функция создает пункт <OPTION> для выпадающего списка.

Входящие параметры:

- CVALUE (VARCHAR2): Текст для пункта списка.
- CSELECTED (VARCHAR2): Если значение не NULL, добавляет атрибут SELECTED.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formselectoption('RU', 'selected', 'id="Россия"');
```

HTF_FORMSUBMIT

Функция создает кнопку, отправляющую форму на сервер.

Входящие параметры:

- CNAME (VARCHAR2): Значение для атрибута NAME.
- CVALUE (VARCHAR2): Значение для атрибута VALUE (текст на кнопке).
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_formsubmit('submit_order', 'Отправить заказ', 'class="btn-primary"
id="submitBtn"');
```

HTF_FORMTEXT

Эта функция генерирует тег <INPUT> с типом TYPE="text", который создает поле для одной строки текста.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- csize (VARCHAR2 DEFAULT NULL) – Значение атрибута SIZE.
- cmaxlength (VARCHAR2 DEFAULT NULL) – Значение атрибута MAXLENGTH.
- cvalue (VARCHAR2 DEFAULT NULL) – Значение атрибута VALUE.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <INPUT> с параметром TYPE="text" в HTML документе.

Примеры запросов:

```
select htf_formtext('txt_label', '18', '20', 'empty label', 'myAttr=value');
```

HTF_FORMTEXTAREA

Эта функция генерирует тег <TEXTAREA>, который создает текстовое поле, не содержащее предопределенного текста. Это поле позволяет вводить несколько строк текста.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htf_formtextarea('txt_area_2', 3, 3, 'center', 'myAttr=value');
```

HTF_FORMTEXTAREA2

Эта функция генерирует тег <TEXTAREA>, который создает текстовое поле, не содержащее предопределенного текста. Это поле позволяет вводить несколько строк текста. В отличие от функции HTF_FORMTEXTAREA, имеет параметр cwrap, позволяющий указать стиль переноса текста.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cwrap (VARCHAR2 DEFAULT NULL) – Значение атрибута WRAP.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htf_formtextarea2('txt_area_2', 3, 3, 'center', 'hard', 'myAttr=value');
```

HTF_FORMTEXTAREAOPEN

Эта функция генерирует тег <TEXTAREA>, который отмечает начало текстового поля формы. Конец текстового поля формы отмечается с помощью процедуры HTF_FORMTEXTAREACLOSE.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htf_formtextareaopen('txt_area_3', 2, 2, 'center', 'myAttr=value');
```

HTF_FORMTEXTAREAOPEN2

Эта функция генерирует тег <TEXTAREA>, который отмечает начало текстового поля формы. В отличие от функции HTF_FORMTEXTAREAOPEN, имеет параметр cwrap, позволяющий указать стиль переноса текста. Конец текстового поля формы отмечается с помощью процедуры HTF_FORMTEXTAREACLOSE.

Входящие параметры:

- cname (VARCHAR2) – Значение атрибута NAME.
- nrows (NUMBER) – Значение атрибута ROWS.
- ncolumns (NUMBER) – Значение атрибута COLUMNS.
- calign (VARCHAR2 DEFAULT NULL) – Значение атрибута ALIGN.
- cwrap (VARCHAR2 DEFAULT NULL) – Значение атрибута WRAP.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <TEXTAREA> в HTML документе.

Примеры запросов:

```
select htf_formtextareaopen2('txt_area_3', 2, 2, 'center', 'hard', 'myAttr=value');
```

HTF_FRAME

Функция создает тег <FRAME> для определения отдельного фрейма в наборе фреймов.

Входящие параметры:

- CSRC (VARCHAR2): URL для отображения во фрейме.
- CNAME (VARCHAR2): Значение для атрибута NAME.
- CMARGINWIDTH (VARCHAR2): Значение для атрибута MARGINWIDTH.
- CMARGINHEIGHT (VARCHAR2): Значение для атрибута MARGINHEIGHT.

- CSCROLLING (VARCHAR2): Значение для атрибута SCROLLING ("yes", "no", "auto").
- CNORESIZE (VARCHAR2): Если значение этого параметра не NULL, в тег добавляется атрибут NORESIZE.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_frame('/maps/main.html', 'main_frame', '10', '10', 'auto', '1',
'class="content-frame" id="main"');
```

HTF_FRAMESETOPEN

Функция создает открывающий тег <FRAMESET> для разбиения окна на фреймы.

Входящие параметры:

- CROWS (VARCHAR2): Значение для атрибута ROWS.
- CCOLS (VARCHAR2): Значение для атрибута COLS.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_framesetopen('20%,80%', '25%,50%,25%', 'border="0" frameborder="0"');
```

HTF_HEADER

Функция создает HTML-заголовок указанного уровня с возможностью выравнивания и дополнительных атрибутов.

Входящие параметры:

- NSIZE (NUMBER): Уровень заголовка (1-6).
- CHEADER (VARCHAR2): Текст заголовка.
- CALIGN (VARCHAR2): Значение для атрибута ALIGN.
- CNOWRAP (VARCHAR2): Значение для атрибута NOWRAP.
- CCLEAR (VARCHAR2): Значение для атрибута CLEAR.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_header(4, 'Квартальный отчёт', 'right', 'nowrap', 'no', 'class="section-
title" style="color:#2c3e50;"');
```

HTF_HR

Функция вставляет горизонтальную разделительную линию с возможностью пользовательского изображения.

Входящие параметры:

- CCLEAR (VARCHAR2): Значение для атрибута CLEAR.
- CSRC (VARCHAR2): Значение для атрибута SRC (пользовательское

изображение).

- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_hr('all', '/images/separator.gif', 'class="hr-divider" style="border:
none;"');
```

HTF_HTITLE

Функция создает заголовок с настраиваемыми атрибутами (устаревшая, рекомендуется HEADER).

Входящие параметры:

- CTITLE (VARCHAR2): Текст заголовка.
- NSIZE (NUMBER): Уровень заголовка.
- CALIGN (VARCHAR2): Выравнивание.
- CNOWRAP (VARCHAR2): Запрет переноса.
- CCLEAR (VARCHAR2): Значение для атрибута CLEAR.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_htitle('Добро пожаловать на наш сайт', 2, 'center', 'yes', 'none',
'class="main-title" style="color:#2c3e50;"');
```

HTF_IMG

Функция создает тег изображения с поддержкой карт встраивания.

Входящие параметры:

- CURL (VARCHAR2): Значение для атрибута SRC (URL изображения).
- CALIGN (VARCHAR2): Значение для атрибута ALIGN.
- CALT (VARCHAR2): Значение для атрибута ALT (альтернативный текст).
- CISMAP (VARCHAR2): Если значение не NULL, добавляет атрибут ISMAP.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_img('/images/logo.png', 'center', 'Логотип компании', 'yes', 'class="logo"
style="border:0;"');
```

HTF_IMG2

Функция создает тег изображения с поддержкой карт встраивания.

Входящие параметры:

- CURL (VARCHAR2): Значение для атрибута SRC (URL изображения).
- CALIGN (VARCHAR2): Значение для атрибута ALIGN.
- CALT (VARCHAR2): Значение для атрибута ALT (альтернативный текст).
- CISMAP (VARCHAR2): Если значение не NULL, добавляет атрибут ISMAP.

- CUSEMAP (VARCHAR2): Значение для атрибута USEMAP (клиентская карта).
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты, которые будут включены в тег как есть.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_img2('/images/map.png', 'left', 'Интерактивная карта офиса', NULL,
'#office_map', 'class="map-image" style="border:2px solid #ccc;"');
```

HTF_ISINDEX

Функция создает простое поле поиска (устаревший тег <ISINDEX>).

Входящие параметры:

- CPROMPT (VARCHAR2): Значение для атрибута PROMPT.
- CURL (VARCHAR2): Значение для атрибута HREF.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_isindex('Enter request for search', '/cgi-bin/search.pl');
```

HTF_ITALIC

Функция создает теги курсивного выделения текста.

Входящие параметры:

- CTEXT (VARCHAR2): Текст для отображения курсивом.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_italic('Обратите внимание на этот важный термин', 'class="emphasis"
style="color:#0066cc;"');
```

HTF_KBD

Функция форматирует текст как клавиатурный ввод моноширинным шрифтом.

Входящие параметры:

- CTEXT (VARCHAR2): Текст для отображения моноширинным шрифтом.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_kbd('Ctrl+S', 'class="keyboard" style="background:#f4f4f4; padding:2px 6px;
border:1px solid #ccc; border-radius:3px;"');
```

HTF_KEYBOARD

Функция форматирует текст как клавиатурный ввод (синоним KBD).

Входящие параметры:

- CTEXT (VARCHAR2): Текст для отображения моноширинным шрифтом.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_keyboard('Ctrl+S', 'class="keyboard" style="background:#f4f4f4; padding:2px
6px; border:1px solid #ccc; border-radius:3px;"');
```

HTF_LINE

Функция создает горизонтальную линию (синоним HR).

Входящие параметры:

- CCLEAR (VARCHAR2): Значение для атрибута CLEAR.
- CSRC (VARCHAR2): Значение для атрибута SRC.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_line('both', '/images/divider.gif', 'class="separator" style="border: none;
height: 2px;"');
```

HTF_LINKREL

Функция создает тег <LINK> для определения связи между документами (например, для CSS).

Входящие параметры:

- CREL (VARCHAR2): Значение для атрибута REL (отношение от текущего к целевому).
- CURL (VARCHAR2): Значение для атрибута HREF.
- CTITLE (VARCHAR2): Значение для атрибута TITLE.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_linkrel('stylesheet', '/css/main.css', 'Основной стиль');
```

HTF_LINKREV

Функция создает тег <LINK> с обратным отношением между документами.

Входящие параметры:

- CREV (VARCHAR2): Значение для атрибута REV (отношение от целевого к текущему).
- CURL (VARCHAR2): Значение для атрибута HREF.
- CTITLE (VARCHAR2): Значение для атрибута TITLE.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_linkrev('prev', '/chapter1.html', 'Предыдущая глава');
```

HTF_LISTHEADER

Функция создает заголовок списка (списки определений).

Входящие параметры:

- CTEXT (VARCHAR2): Текст заголовка списка.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_listheader('Основные термины', 'class="list-title" style="font-weight:bold;
color:#2c3e50;"');
```

HTF_LISTITEM

Функция создает элемент маркированного, нумерованного или определительного списка.

Входящие параметры:

- CTEXT (VARCHAR2): Текст элемента списка.
- CCLEAR (VARCHAR2): Значение для атрибута CLEAR.
- CDINGBAT (VARCHAR2): Значение для атрибута DINGBAT.
- CSRC (VARCHAR2): Значение для атрибута SRC.
- CATTRIBUTES (VARCHAR2): Дополнительные атрибуты.

Выходные параметры:

- VARCHAR2 - сгенерированная строка с тегом

Типичные сценарии использования:

- генерация строки с тэгом

Примеры запросов:

```
SELECT htf_listitem('Настройка параметров соединения', 'none', NULL, NULL,
'class="step-item" data-step="1"');
```

HTF_MAPOPEN

Эта функция генерирует тег <MAP>, который отмечает начало набора регионов на карте изображений на стороне клиента. Чтобы отметить конец набора регионов на карте изображений на стороне клиента, используйте процедуру HTF_MAPCLOSE.

Входящие параметры:

- cname (VARCHAR2 DEFAULT NULL) – Значение атрибута NAME.
- cattributes (VARCHAR2 DEFAULT NULL) – другие атрибуты, которые будут включены в тег буквально.

Выходные параметры:

- tag (VARCHAR2) - строка, содержащая тег и дополнительные данные.

Типичные сценарии использования:

- Формирование тега <MAP> в HTML документе.

Примеры запросов:

```
select htf_mapopen('infographic', 'myAttr=value');
```

5.2.37. Пакет UTL_ENCODE

UTL_ENCODE.BASE64_ENCODE

Функция преобразует двоичные данные (RAW) в строку в кодировке Base64.

Входящие параметры:

- r (RAW) – исходные двоичные данные для кодирования.
- charset_id (PLS_INTEGER) – идентификатор кодировки символов для выходной строки (опционально).
- newline (PLS_INTEGER) – флаг для вставки символов новой строки через каждые 76 символов (опционально, например, CHR(13)||CHR(10)).

Выходные параметры:

- Возвращает закодированные данные в формате Base64.

Типичные сценарии использования:

- Подготовка двоичных данных (например, файлов, подписей) для передачи через текстовые протоколы (HTTP, электронная почта).
- Кодирование вложений для писем, отправляемых через UTL_MAIL.
- Хранение двоичных данных в текстовых полях (CLOB, VARCHAR2).

Примеры запросов:

```
select utl_encode_base64_encode('48656C6C6F2E');
```

UTL_ENCODE.BASE64_DECODE

Функция декодирует строку из формата Base64 обратно в исходные двоичные данные (RAW).

Входящие параметры:

- r (RAW) – данные в кодировке Base64 для декодирования.

Выходные параметры:

- Возвращает декодированные двоичные данные.

Типичные сценарии использования:

- Обработка вложений электронной почты, полученных в кодировке Base64.
- Извлечение двоичных данных, хранящихся в текстовом виде.

- Работа с API, которые передают двоичный контент в Base64.

Примеры запросов:

```
select utl_encode_base64_decode('5347567362473875');
```

UTL_ENCODE.UUENCODE

Функция кодирует двоичные данные (RAW) в формат UUENCODE.

Входящие параметры:

- r (RAW) – исходные двоичные данные для кодирования.
- type (PLS_INTEGER) – тип кодирования (например, для указания имени файла и режима доступа в заголовке).

Выходные параметры:

- Возвращает закодированные данные в формате UUENCODE.

Типичные сценарии использования:

- Обработка устаревших систем или данных, где до сих пор используется UUENCODE.
- Совместимость с протоколами, исторически применявшими это кодирование.

Примеры запросов:

```
DO $$
DECLARE
    src_text TEXT := 'Привет мир';
    raw_data RAW;
    encoded_raw RAW;
    decoded_raw RAW;
    decoded_text TEXT;
BEGIN
    -- Пытаемся преобразовать bytea в RAW
    raw_data := convert_to(src_text, 'UTF8')::RAW;

    encoded_raw := utl_encode_uuencode(raw_data, NULL, NULL, NULL);
    decoded_raw := utl_encode_uudecode(encoded_raw);

    -- Пытаемся преобразовать RAW обратно в bytea, а затем в текст
    decoded_text := convert_from(decoded_raw::bytea, 'UTF8');

    RAISE NOTICE 'Исходный текст: %', src_text;
    RAISE NOTICE 'UUEncoded (RAW): %', encoded_raw;
    RAISE NOTICE 'Расшифровано: %', decoded_text;
END;
$$;
```

UTL_ENCODE.TEXT_DECODE

Функция декодирует текст из заданной кодировки символов (character set) в RAW.

Входящие параметры:

- buf (VARCHAR2) – текст для декодирования.
- charset (VARCHAR2) – исходная кодировка символов текста.

Выходные параметры:

- Возвращает декодированные двоичные данные.

Типичные сценарии использования:

- Преобразование текстовых данных, полученных из внешних источников с различными кодировками, в единый внутренний формат.
- Подготовка текста для криптографических операций, которые требуют RAW-данных.

Примеры запросов:

```
select utl_encode_text_decode('0J/RgNC40LLQtdGC', 'UTF8', 1);
```

UTL_ENCODE.QUOTED_PRINTABLE_DECODE

Функция декодирует данные из формата Quoted-Printable в исходный RAW.

Входящие параметры:

- buf (RAW) – данные в кодировке Quoted-Printable.

Выходные параметры:

- Возвращает декодированные двоичные данные.

Типичные сценарии использования:

- Обработка содержимого электронных писем, закодированных методом Quoted-Printable.
- Декодирование текстовых данных, содержащих символы, не входящие в ASCII.

Примеры запросов:

```
select utl_encode_quoted_printable_decode('48656C6C6F203D45323D38323D4143');
```

UTL_ENCODE.QUOTED_PRINTABLE_ENCODE

Функция кодирует двоичные данные (RAW) в формат Quoted-Printable.

Входящие параметры:

- buf (RAW) – исходные двоичные данные для кодирования.

Выходные параметры:

- Возвращает закодированные данные в формате Quoted-Printable.

Типичные сценарии использования:

- Подготовка текста электронного письма для корректной передачи через

SMTP, особенно для не-ASCII символов.

- Кодирование данных перед вставкой в почтовые заголовки.

Примеры запросов:

```
select utl_encode_quoted_printable_encode('48656C6C6F20E282AC');
```

UTL_ENCODE.MIMEHEADER_DECODE

Функция декодирует строку из MIME-заголовка электронного письма (например, Subject, From), которая может быть закодирована для поддержки не-ASCII символов.

Входящие параметры:

- buf (VARCHAR2) – закодированная строка заголовка.

Выходные параметры:

- Возвращает декодированную строку.

Типичные сценарии использования:

- Корректное отображение тем писем и имен отправителей на разных языках.
- Парсинг входящих электронных писем.

Примеры запросов:

```
select utl_encode_mimeheader_decode('=?UTF-8?B?0J/RgNC40LLQtdGC?=');
```

UTL_ENCODE_BASE64_DECODE_INTERNAL

Вспомогательная функция для: utl_encode_base64_decode

Описание: декодирует данные из base64.

Входящие параметры:

- str (RAW) – данные в формате base64
- onempty (BOOLEAN) – поведение при пустых данных

Выходные параметры:

- RAW – декодированные данные

Типичные сценарии использования:

- Декодирование base64-строк
- Преобразование закодированных данных

Примеры запросов:

```
SELECT utl_encode_base64_decode_internal('...', false);
```

UTL_ENCODE_GET_VERSION

Функция возвращает номер версии пакета UTL_ENCODE для кодирования данных (Base64, Quoted-Printable и др.).

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для кодирования
- Совместимость форматов

Примеры запросов:

```
SELECT utl_encode_get_version();
```

5.2.38. Пакет UTL_INNADR

UTL_INADDR.GET_HOST_NAME;

Функция возвращает имя хоста для заданного IP-адреса (DNS reverse lookup).

Входящие параметры:

- ip (VARCHAR2) – IP-адрес в формате строки (например, '192.168.1.1').

Выходные параметры:

- Возвращает имя хоста, соответствующее IP-адресу. Если преобразование невозможно, возвращается исходный IP.

Типичные сценарии использования:

- Определение имени машины-источника соединения или запроса для целей аудита и безопасности.
- Логирование не только IP, но и понятных имен серверов в приложениях.
- Валидация IP-адресов путем проверки их регистрации в DNS.

Примеры запросов:

```
SELECT utl_inaddr_get_host_name('127.0.0.1');
```

UTL_INADDR.GET_HOST_ADDRESS;

Функция возвращает IP-адрес для заданного имени хоста (DNS lookup).

Входящие параметры:

- hostname (VARCHAR2) – имя хоста.

Выходные параметры:

- Возвращает IP-адрес, соответствующий имени хоста. Если преобразование невозможно, возникает исключение.

Типичные сценарии использования:

- Преобразование доменных имен в IP-адреса для сетевых операций (например, настройка ACL, проверка доступности).
- Определение адреса целевого сервера перед установкой сетевого соединения из PL/SQL.
- Кэширование IP-адресов для уменьшения задержек, связанных с DNS.

Примеры запросов:

```
SELECT utl_inaddr_get_host_address('localhost');
```

```
SELECT utl_inaddr_get_host_address('google.com');
```

UTL_INADDR_GET_LOCAL_IP

Вспомогательная функция для: utl_inaddr_get_host_address

Описание: возвращает локальный IP-адрес.

Входящие параметры:

- Нет параметров

Выходные параметры:

- TEXT – локальный IP-адрес

Типичные сценарии использования:

- Определение сетевого адреса сервера
- Конфигурация сетевых приложений

Примеры запросов:

```
SELECT utl_inaddr_get_local_ip();
```

UTL_INADDR_GET_VERSION

Функция возвращает номер версии пакета UTL_INADDR для работы с IP-адресами.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для сетевых функций
- Совместимость DNS-запросов

Примеры запросов:

```
SELECT utl_inaddr_get_version();
```

5.2.39. Пакет UTL_MATCH

UTL_MATCH.EDIT_DISTANCE

Функция вычисляет редакционное расстояние (расстояние Левенштейна) между двумя строками. Возвращает минимальное количество операций вставки, удаления и замены символа, необходимых для превращения одной строки в другую.

Входящие параметры:

- s1 (VARCHAR2) – первая строка.
- s2 (VARCHAR2) – вторая строка.

Выходные параметры:

- Возвращает редакционное расстояние (0 для идентичных строк, больше 0 для различных).

Типичные сценарии использования:

- Поиск неточных совпадений при очистке данных (например, устранение опечаток в именах).
- Сопоставление записей из разных источников, где возможны расхождения в написании.
- Оценка схожести коротких текстовых полей.

Примеры запросов:

```
select utl_match_edit_distance('shackleford','shackelford');
```

UTL_MATCH.EDIT_DISTANCE_SIMILARITY

Функция вычисляет процентную схожесть двух строк на основе редакционного расстояния. Возвращает значение от 0 (совсем не похожи) до 100 (идентичны).

Входящие параметры:

- s1 (VARCHAR2) – первая строка.
- s2 (VARCHAR2) – вторая строка.

Выходные параметры:

- Возвращает процент схожести (0-100).

Типичные сценарии использования:

- Ранжирование результатов нечеткого поиска по степени соответствия.
- Задание пороговых значений для автоматического принятия решений (например, "слить записи, если схожесть > 90%").
- Визуализация степени отличия строк для пользователя.

Примеры запросов:

```
select utl_match_edit_distance_similarity(NULL, NULL);
```

UTL_MATCH.JARO_WINKLER

Функция вычисляет схожесть двух строк по алгоритму Джаро-Винклера. Алгоритм учитывает общие символы и их позиции, давая более высокий вес совпадениям в начале строк. Возвращает значение от 0 до 1.

Входящие параметры:

- s1 (VARCHAR2) – первая строка.
- s2 (VARCHAR2) – вторая строка.

Выходные параметры:

- Возвращает коэффициент схожести по Джаро-Винклеру.

Типичные сценарии использования:

- Сравнение имен людей, названий компаний, адресов, где важен порядок и

начало строки.

- Использование в системах дедупликации данных о клиентах или контрагентах.
- Комбинирование с другими метриками для повышения точности нечеткого сопоставления.

Примеры запросов:

```
select utl_match_jaro_winkler('hello','world');
```

UTL_MATCH.JARO_WINKLER_SIMILARITY

Функция вычисляет схожесть по Джаро-Винклеру и возвращает результат в виде целого числа от 0 до 100 (процент).

Входящие параметры:

- s1 (VARCHAR2) – первая строка.
- s2 (VARCHAR2) – вторая строка.

Выходные параметры:

- Возвращает процент схожести по Джаро-Винклеру (0-100).

Типичные сценарии использования:

- Те же сценарии, что и для JARO_WINKLER, но с выводом в более привычном процентном формате для отчетов или пороговых проверок.

Примеры запросов:

```
select utl_match_jaro_winkler_similarity('shackleford','shackelford');
```

UTL_MATCH_LEVENSHTEIN

Вспомогательная функция для: utl_match_edit_distance, utl_match_edit_distance_similarity

Описание: вычисляет расстояние Левенштейна между строками.

Входящие параметры:

- a (TEXT) – первая строка
- b (TEXT) – вторая строка

Выходные параметры:

- INT4 – расстояние Левенштейна

Типичные сценарии использования:

- Оценка схожести строк
- Поиск нечетких совпадений

Примеры запросов:

```
SELECT utl_match_levenshtein('kitten','sitting');
```

UTL_MATCH_GET_VERSION

Функция возвращает номер версии пакета UTL_MATCH для сравнения строк.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии алгоритмов сравнения
- Совместимость метрик

Примеры запросов:

```
SELECT utl_match_get_version();
```

5.2.40. Пакет UTL_RAW

UTL_RAW.BIT_AND

Функция выполняет побитовую операцию "И" (AND) над двумя значениями RAW.

Входящие параметры:

- r1 (RAW) – первое значение.
- r2 (RAW) – второе значение.

Выходные параметры:

- Возвращает результат побитового AND.

Типичные сценарии использования:

- Применение битовых масок к двоичным данным (например, извлечение определенных флагов из байта состояния).
- Низкоуровневая обработка сетевых пакетов или протокольных заголовков.
- Реализация криптографических алгоритмов или операций с хешами.

Примеры запросов:

```
SELECT utl_raw_bit_and('\xFF00', '\x00FF') AS result;
```

UTL_RAW.BIT_OR

Функция выполняет побитовую операцию "ИЛИ" (OR) над двумя значениями RAW.

Входящие параметры:

- r1 (RAW) – первое значение.
- r2 (RAW) – второе значение.

Выходные параметры:

- Возвращает результат побитового OR.

Типичные сценарии использования:

- Установка определенных битов в двоичных данных (например, включение флагов).

- Комбинирование масок или битовых полей.

Примеры запросов:

```
SELECT utl_raw_bit_or('\xFF00', '\x00FF') AS result;
```

UTL_RAW.BIT_XOR

Функция выполняет побитовую операцию "исключающее ИЛИ" (XOR) над двумя значениями RAW.

Входящие параметры:

- r1 (RAW) – первое значение.
- r2 (RAW) – второе значение.

Выходные параметры:

- Возвращает результат побитового XOR.

Типичные сценарии использования:

- Простое шифрование/дешифрование данных с использованием ключа (шифр Вернама).
- Проверка четности, вычисление контрольных сумм.
- Сравнение данных: результат XOR будет нулевым для идентичных значений.

Примеры запросов:

```
SELECT utl_raw_bit_xor('\xFFFF', '\x0000') AS result;
```

UTL_RAW.BIT_COMPLEMENT

Функция выполняет побитовую операцию "НЕ" (дополнение) над значением RAW, инвертируя все биты.

Входящие параметры:

- r (RAW) – исходное значение.

Выходные параметры:

- Возвращает инвертированное значение.

Типичные сценарии использования:

- Инверсия битовых масок.
- Преобразование данных в рамках специфических алгоритмов или протоколов.

Примеры запросов:

```
SELECT utl_raw_bit_complement(utl_raw_bit_complement('\xA1')) AS double_complement;
```

UTL_RAW.CAST_FROM_BINARY_DOUBLE

Функция преобразует значение типа BINARY_DOUBLE в тип RAW.

Входящие параметры:

- b (BINARY_DOUBLE) – значение с плавающей точкой двойной точности.

Выходные параметры:

- Возвращает двоичное представление числа.

Типичные сценарии использования:

- Подготовка числовых данных с плавающей точкой для криптографических операций, которые требуют RAW-ввода.
- Сохранение или передача чисел в точном двоичном формате без потерь.
- Низкоуровневая работа с данными, где важно внутреннее представление числа.

Примеры запросов:

```
SELECT utl_raw_cast_from_binary_double(123.45::binary_double, 1::binary_integer) AS
result;
```

UTL_RAW.CAST_TO_BINARY_DOUBLE

Функция преобразует значение типа RAW в тип BINARY_DOUBLE.

Входящие параметры:

- r (RAW) – двоичные данные, представляющие число BINARY_DOUBLE.

Выходные параметры:

- Возвращает восстановленное число с плавающей точкой.

Типичные сценарии использования:

- Восстановление чисел после криптографических операций или передачи в RAW-формате.
- Чтение двоичных данных, полученных из внешних систем, где числа хранятся в стандартном IEEE формате.

Примеры запросов:

```
SELECT utl_raw_cast_to_binary_double('\x405E666666666666', 1::binary_integer) AS
result;
```

UTL_RAW.CAST_FROM_BINARY_FLOAT

Функция преобразует значение типа BINARY_FLOAT в тип RAW.

Входящие параметры:

- b (BINARY_FLOAT) – значение с плавающей точкой одинарной точности.

Выходные параметры:

- Возвращает двоичное представление числа.

Типичные сценарии использования:

- Аналогично CAST_FROM_BINARY_DOUBLE, но для чисел одинарной точности.
- Оптимизация использования памяти/пропускной способности при работе с большими массивами чисел.

Примеры запросов:

```
SELECT utl_raw_cast_from_binary_float(123.45::binary_float, 1::binary_integer) AS
result;
```

UTL_RAW.CAST_TO_BINARY_FLOAT

Функция преобразует значение типа RAW в тип BINARY_FLOAT.

Входящие параметры:

- r (RAW) – двоичные данные, представляющие число BINARY_FLOAT.

Выходные параметры:

- Возвращает восстановленное число с плавающей точкой.

Типичные сценарии использования:

- Восстановление чисел одинарной точности из их двоичного представления.

Примеры запросов:

```
SELECT utl_raw_cast_to_binary_float('\x42F670A4', 1::binary_integer) AS result;
```

UTL_RAW.CAST_FROM_BINARY_INTEGER

Функция преобразует значение типа BINARY_INTEGER или PLS_INTEGER в тип RAW.

Входящие параметры:

- b (BINARY_INTEGER) – целочисленное значение.

Выходные параметры:

- Возвращает двоичное представление числа.

Типичные сценарии использования:

- Преобразование целых чисел для операций с RAW-данными (хеширование, шифрование).
- Подготовка параметров для низкоуровневых API или пользовательских агрегатных функций.

Примеры запросов:

```
SELECT utl_raw_cast_from_binary_integer(123::binary_integer, 1::binary_integer) AS
result;
```

UTL_RAW.CAST_TO_BINARY_INTEGER

Функция преобразует значение типа RAW в тип BINARY_INTEGER.

Входящие параметры:

- r (RAW) – двоичные данные, представляющие целое число.

Выходные параметры:

- Возвращает восстановленное целое число.

Типичные сценарии использования:

- Восстановление целочисленных параметров после обработки в RAW-формате.
- Извлечение числовых значений из двоичных структур данных.

Примеры запросов:

```
SELECT utl_raw_cast_to_binary_integer('\x0000007B', 1::binary_integer) AS result;
```

UTL_RAW.CAST_TO_VARCHAR2

Функция преобразует значение типа RAW в строку VARCHAR2. Каждый байт RAW интерпретируется как символ в кодировке базы данных (чаще всего это однобайтовая кодировка для RAW ⇐ 2000 байт).

Входящие параметры:

- r (RAW) – двоичные данные.

Выходные параметры:

- Возвращает строку, полученную прямым отображением байтов в символы.

Типичные сценарии использования:

- Быстрое преобразование RAW-данных, гарантированно содержащих только

однобайтовые символы (например, шестнадцатеричные строки, Base64, коды ASCII).

- Отладка и логирование содержимого RAW-переменных в читаемом виде (если содержимое - текст).

Примеры запросов:

```
SELECT utl_raw_cast_to_varchar2(convert_to('Привет мир', 'UTF8')::RAW) AS result;
```

UTL_RAW.CAST_TO_RAW

Функция преобразует строку VARCHAR2 в тип RAW. Каждый символ строки преобразуется в байт на основе кодировки базы данных. Важно: Подходит только для строк, содержащих символы из однобайтового набора.

Входящие параметры:

- c (VARCHAR2) – исходная строка.

Выходные параметры:

- Возвращает двоичные данные.

Типичные сценарии использования:

- Преобразование текстовых строк, содержащих только символы Latin-1 или ASCII, в RAW для дальнейшей обработки.
- Создание тестовых или константных RAW-значений из строковых литералов.

Примеры запросов:

```
SELECT utl_raw_cast_to_raw('Привет мир') AS result;
```

UTL_RAW.COMPARE

Функция сравнивает два значения RAW. Возвращает 0, если они идентичны. Если отличаются, возвращает позицию первого отличающегося байта (начиная с 1). Если одно значение является префиксом другого, возвращает длину более короткого значения + 1.

Входящие параметры:

- r1 (RAW) – первое значение.
- r2 (RAW) – второе значение.

Выходные параметры:

- Возвращает результат сравнения (0 – равны, >0 – позиция первого отличия).

Типичные сценарии использования:

- Побайтовое сравнение двоичных данных (хешей, зашифрованных блоков, сериализованных объектов).
- Определение, является ли одно значение началом другого (префиксная проверка).

Примеры запросов:

```
SELECT
    utl_raw_compare(
        convert_to('Привет мир', 'UTF8')::RAW,
        convert_to('Привет мир', 'UTF8')::RAW,
        '\x00'
    ) AS result;
```

UTL_RAW.CONCAT

Функция объединяет (конкатенирует) несколько значений RAW в одно.

Входящие параметры:

- r1 (RAW) – первое значение.
- r2 (RAW) – второе значение.
- r3 … rN (RAW) – дополнительные значения (до 12).

Выходные параметры:

- Возвращает результат конкатенации всех аргументов.

Типичные сценарии использования:

- Сборка двоичных пакетов или сообщений из отдельных частей (заголовок + тело + подпись).

- Построение ключей или входных данных для криптографических функций путем объединения нескольких элементов.
- Динамическое формирование двоичного содержимого для вложений или BLOB.

Примеры запросов:

```
SELECT
    utl_raw_concat(
        convert_to('Привет', 'UTF8')::RAW,
        convert_to(' ', 'UTF8')::RAW,
        convert_to('мир', 'UTF8')::RAW,
        ''::RAW, ''::RAW, ''::RAW, ''::RAW, ''::RAW, ''::RAW, ''::RAW, ''::RAW,
''::RAW
    ) AS result;
```

UTL_RAW.COPIES

Функция возвращает значение RAW, состоящее из указанного количества повторений исходного значения RAW.

Входящие параметры:

- r (RAW) – шаблон для повторения.
- n (PLS_INTEGER) – количество повторений.

Выходные параметры:

- Возвращает сконкатенированные копии исходного значения.

Типичные сценарии использования:

- Создание тестовых данных заданного размера (например, заполнителей, паттернов).
- Генерация константных масок или буферов для инициализации.

Примеры запросов:

```
SELECT utl_raw_copies(convert_to('Привет мир', 'UTF8')::RAW, 3) AS result;
```

UTL_RAW.LENGTH

Функция возвращает длину значения RAW в байтах.

Входящие параметры:

- r (RAW) – исходное значение.

Выходные параметры:

- Возвращает длина в байтах.

Типичные сценарии использования:

- Проверка размера двоичных данных перед их обработкой, передачей или сохранением.
- Валидация входных параметров (например, длина хеша должна быть 32 байта).
- Выделение памяти или буферов на основе размера данных.

Примеры запросов:

```
SELECT utl_raw_length(convert_to('Привет мир', 'UTF8')::RAW) AS result;
```

UTL_RAW.OVERLAY

Функция заменяет часть целевого значения RAW, начиная с указанной позиции, на указанное замещающее значение RAW.

Входящие параметры:

- target (RAW) – целевое значение, которое будет модифицировано.
- overlay (RAW) – замещающее значение.
- position (PLS_INTEGER) – позиция в целевом значении, с которой начинается замена (по умолчанию 1).
- length (PLS_INTEGER) – количество байт в целевом значении, которые нужно заменить (по умолчанию длина замещающего значения).

Выходные параметры:

- Возвращает модифицированное значение.

Типичные сценарии использования:

- Модификация двоичных структур данных (например, обновление поля в заголовке файла или пакета).
- "Исправление" или маскирование определенных участков в двоичных данных.
- Вставка или замена блоков в потоке данных.

Примеры запросов:

```
SELECT
    utl_raw_overlay(
        convert_to('человек!', 'UTF8')::RAW,      -- RAW для наложения
        convert_to('Привет, мир!', 'UTF8')::RAW, -- исходный RAW
        15::binary_integer,                       -- позиция начала (со 15-го байта)
        NULL,                                     -- количество байтов для замены,
        NULL
    ) AS result;
```

UTL_RAW.REVERSE

Функция возвращает значение RAW, в котором порядок байтов обратен исходному.

Входящие параметры:

- r (RAW) – исходное значение.

Выходные параметры:

- Возвращает значение с обратным порядком байтов.

Типичные сценарии использования:

- Конвертация порядка байтов (endianness) между системами (little-endian <→ big-endian).
- Обработка данных, записанных в обратном порядке.
- Специфические алгоритмы, требующие обращения последовательности байт.

Примеры запросов:

```
SELECT utl_raw_reverse(convert_to('Привет мир', 'UTF8')::RAW) AS result;
```

UTL_RAW.SUBSTR

Функция извлекает подстроку (подпоследовательность байтов) из значения RAW.

Входящие параметры:

- r (RAW) – исходное значение.
- pos (PLS_INTEGER) – начальная позиция для извлечения (по умолчанию 1).
- len (PLS_INTEGER) – количество извлекаемых байт. Если не указано или превышает доступное количество, извлекаются все байты до конца.

Выходные параметры:

- Возвращает извлеченная часть.

Типичные сценарии использования:

- Извлечение отдельных полей из двоичной структуры (например, первых 4 байт как заголовка).
- Разбор входящих двоичных сообщений или файлов.
- Получение части хеш-значения или подписи.

Примеры запросов:

```
SELECT utl_raw_substr(convert_to('Привет мир', 'UTF8')::RAW, 1, 13) AS result;
```

UTL_RAW.TRANSLATE

Функция преобразует значение RAW, заменяя каждый байт, найденный в строке "from_set", на соответствующий байт из строки "to_set". Аналог TRANSLATE для строк, но для байтов.

Входящие параметры:

- r (RAW) – исходное значение.
- from_set (RAW) – набор байтов для поиска.

- to_set (RAW) – набор байтов для замены.

Выходные параметры:

- Возвращает преобразованное значение.

Типичные сценарии использования:

- Простая замена или маскирование определенных байтов в двоичных данных (например, удаление нулевых байтов).
- Применение простых шифров замены (substitution cipher) на уровне байтов.
- Нормализация данных (приведение разных кодов одного символа к единому виду).

Примеры запросов:

```
SELECT
    utl_raw_translate(
        convert_to('Привет мир', 'UTF8')::RAW,
        convert_to('р', 'UTF8')::RAW,
        convert_to('с', 'UTF8')::RAW
    ) AS result; -- байты меняет
```

UTL_RAW.TRANSLITERATE

Функция преобразует значение RAW из одной кодировки символов в другую. Это побайтовая операция, которая может менять длину результата, если кодировки имеют разный размер символа (например, однобайтовая в UTF-8).

Входящие параметры:

- src (RAW) – исходные двоичные данные.
- src_charset (VARCHAR2) – исходная кодировка.
- dst_charset (VARCHAR2) – целевая кодировка.

Выходные параметры:

- RAW – преобразованные данные в целевой кодировке.

Типичные сценарии использования:

- Конвертация содержимого файлов или BLOB между различными

кодировками на уровне байтов.

- Подготовка текстовых данных для систем, требующих конкретной кодировки.
- Исправление проблем с "кракозябрами", когда данные были сохранены в неверной кодировке.

Примеры запросов:

```
SELECT utl_raw_transliterate('\x123456','\xAABB','\x123456', '\xFF') AS result;
```

UTL_RAW.XRANGE

Функция возвращает значение RAW, представляющее последовательность байтов от начального до конечного значения включительно.

Входящие параметры:

- start_byte (RAW) – начальный байт (один байт, например, HEXTORAW('00')).
- end_byte (RAW) – конечный байт (один байт, например, HEXTORAW('FF')).

Выходные параметры:

- Возвращает последовательность байтов от start_byte до end_byte.

Типичные сценарии использования:

- Генерация тестовых данных, содержащих все возможные байтовые значения.
- Создание таблиц подстановки или масок для алгоритмов обработки.
- Инициализация буферов последовательными значениями.

Примеры запросов:

```
SELECT utl_raw_xrange('\x01', '\x14') AS result;
```

UTL_RAW.CAST_FROM_NUMBER

Функция преобразует числовое значение в представление типа `RAW`.

Входящие параметры:

- n (NUMBER) – числовое значение, которое необходимо преобразовать в RAW.

Выходные параметры:

- Возвращает значение типа RAW, соответствующее входному числу.

Типичные сценарии использования:

- Преобразование чисел в бинарное представление.
- Подготовка данных для низкоуровневой обработки.
- Использование в сценариях сериализации значений.

Примеры запросов:

```
SELECT utl_raw_cast_from_number(123::number) AS result;
```

UTL_RAW.CAST_TO_NUMBER

Функция преобразует значение типа RAW в число.

Входящие параметры:

- r (RAW) – бинарное значение, которое необходимо интерпретировать как число.

Выходные параметры:

- Возвращает числовое значение, полученное из входного RAW.

Типичные сценарии использования:

- Восстановление чисел из бинарного представления.
- Обработка ранее сериализованных числовых данных.
- Использование в низкоуровневых преобразованиях данных.

Примеры запросов:

```
SELECT utl_raw_cast_to_number('\xC30218') AS result;
```

UTL_RAW.CAST_TO_NVARCHAR2

Функция преобразует значение типа RAW в строку NVARCHAR2.

Входящие параметры:

- r (RAW) – бинарное значение, содержащее текстовые данные.

Выходные параметры:

- Возвращает строковое значение типа NVARCHAR2.

Типичные сценарии использования:

- Преобразование бинарных данных в строку.
- Восстановление текстовых значений из RAW.
- Работа с данными в национальных кодировках.

Примеры запросов:

```
SELECT utl_raw_cast_to_nvarchar2(convert_to('Привет мир', 'UTF8')::RAW) AS result;
```

UTL_RAW.CONVERT

Функция преобразует бинарные данные из одной кодировки в другую.

Входящие параметры:

- r (RAW) – исходные бинарные данные.
- to_charset (VARCHAR2) – целевая кодировка.
- from_charset (VARCHAR2) – исходная кодировка.

Выходные параметры:

- Возвращает значение RAW, перекодированное в указанную целевую кодировку.

Типичные сценарии использования:

- Перекодирование бинарных строк между различными кодировками.
- Подготовка данных для передачи между системами.
- Преобразование текстовых данных, представленных в RAW.

Примеры запросов:

```
SELECT
    utl_raw_convert(
        convert_to('Привет мир', 'ISO_8859_5')::RAW,
        'UTF8',
        'ISO_8859_5'
    ) AS result;
```

UTL_RAW_MAKE_RAW

Создаёт RAW-массив заданной длины.

Входящие параметры:

- LEN (INT8) – длина создаваемого RAW-массива в байтах

Выходные параметры:

- RAW – созданный RAW-массив указанной длины

Типичные сценарии использования:

- Создание буферов для битовых операций
- Используется в UTL_RAW.BIT_AND, UTL_RAW.BIT_OR, UTL_RAW.BIT_XOR, UTL_RAW.BIT_COMPLEMENT, UTL_RAW.CAST_FROM_BINARY_DOUBLE, UTL_RAW.CAST_FROM_BINARY_FLOAT, UTL_RAW.CAST_FROM_BINARY_INTEGER, UTL_RAW.CAST_TO_RAW, UTL_RAW.CONCAT, UTL_RAW.COPIES, UTL_RAW.OVERLAY, UTL_RAW.REVERSE, UTL_RAW.SUBSTR, UTL_RAW.TRANSLATE, UTL_RAW.XRANGE

Примеры запросов:

```
SELECT UTL_RAW_MAKE_RAW(10);
```

UTL_RAW_GET_VERSION

Функция возвращает номер версии пакета UTL_RAW для работы с RAW-данными.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для бинарных операций
- Совместимость конвертации

Примеры запросов:

```
SELECT utl_raw_get_version();
```

5.2.41. Пакет UTL_SMTP

UTL_SMTP.OPEN_CONNECTION

Функция открывает сетевое подключение к указанному SMTP-серверу для последующей отправки электронного письма. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- host (VARCHAR2) – имя хоста или IP-адрес SMTP-сервера.
- port (BINARY_INTEGER) – номер порта SMTP-сервера (по умолчанию 25).
- c (PL/SQL RECORD) – выходная переменная, в которую будет помещён созданный объект подключения (SMTP-коннекция).
- hosts (VARCHAR2) – список хостов для подключения через один из нескольких адресов.
- secure_connection (BOOLEAN) – флаг, указывающий на необходимость установки безопасного (SSL) соединения.

Выходные параметры:

- Возвращает объект подключения типа SMTP_CONNECTION, который используется в других процедурах пакета UTL_SMTP.

Типичные сценарии использования:

- Установка нестандартного порта для подключения к SMTP-серверу.
- Организация отправки писем через сервер, требующий аутентификации (после открытия соединения).
- Настройка безопасного SSL-соединения с почтовым сервером.
- Программная реализация сложного сценария отправки, который не покрывается UTL_MAIL.SEND.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    RAISE INFO 'Result utl_smtp_open_connection: %', v_conn;
    PERFORM utl_smtp_close_connection(v_conn);
END $$;
```

UTL_SMTP.CLOSE_CONNECTION

Процедура закрывает активное SMTP-соединение, установленное ранее с помощью функции OPEN_CONNECTION. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – объект типа SMTP_CONNECTION, представляющий текущее соединение.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Завершение работы с SMTP-сервером после отправки письма.
- Освобождение ресурсов, связанных с активным подключением.
- Корректное закрытие соединения в конце SMTP-сессии.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_close_connection(v_conn);
    RAISE INFO 'Result utl_smtp_close_connection: connection closed';
END $$;
```

UTL_SMTP.QUIT

Процедура отправляет SMTP-команду QUIT серверу для завершения сеанса и закрытия соединения. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Завершение SMTP-сеанса по протоколу перед закрытием соединения.
- Корректное уведомление сервера о завершении работы клиента.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_quit(v_conn);
    RAISE INFO 'Result utl_smtp_quit: session closed';
END $$;
```

UTL_SMTP.NOOP

Процедура отправляет команду NOOP серверу для проверки активности соединения без выполнения других действий. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Проверка того, что соединение с сервером остаётся активным.

- Поддержание сессии при длительной отправке больших писем.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_noop(v_conn);
    RAISE INFO 'Result utl_smtp_noop: NOOP sent';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.HELO

Процедура отправляет SMTP-команду HELO серверу для начала диалога и идентификации клиента по имени хоста. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.
- domain (VARCHAR2) – имя домена клиента, передаваемое серверу.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Инициализация стандартного SMTP-сеанса без расширений.

- Установление базового взаимодействия между клиентом и сервером.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_helo(v_conn, 'example.com'::varchar2);
    RAISE INFO 'Result utl_smtp_helo: HELO example.com sent';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.EHLO

Процедура отправляет SMTP-команду EHLO серверу, идентичную HELO, но с поддержкой расширений протокола ESMTP. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.
- domain (VARCHAR2) – имя домена клиента.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Использование SMTP-расширений, таких как авторизация или передача больших вложений.

- Инициализация безопасных и расширенных SMTP-сеансов.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    RAISE INFO 'Result utl_smtp_ehlo: EHLO example.com sent';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.MAIL

Процедура отправляет SMTP-команду MAIL FROM для указания адреса отправителя письма. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.
- sender (VARCHAR2) – адрес электронной почты отправителя.
- parameters (VARCHAR2) – дополнительные параметры (например, SIZE).

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Определение отправителя письма в SMTP-сеансе.
- Настройка специальных параметров отправки на уровне протокола.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);

    RAISE INFO 'Result utl_smtp_mail: MAIL FROM:<sender@example.com>';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.RCPT

Процедура отправляет SMTP-команду RCPT TO для указания получателя сообщения. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.
- recipient (VARCHAR2) – адрес электронной почты получателя.
- parameters(VARCHAR2) - расширение протокола (SMTP extensions)

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Добавление одного или нескольких получателей письма.
- Указание адресатов при массовой рассылке.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rcpt(v_conn, 'receiver@example.com'::varchar2, NULL::varchar2);

    RAISE INFO 'Result utl_smtp_rcpt: RCPT TO:<receiver@example.com>';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.RSET

Процедура отправляет SMTP-команду RSET серверу для сброса текущего состояния сеанса без закрытия соединения. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Очистка состояния после ошибки отправки.
- Подготовка соединения для нового письма без переоткрытия соединения.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rcpt(v_conn, 'receiver@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rset(v_conn);

    RAISE INFO 'Result utl_smtp_rset: transaction reset';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.DATA

Процедура отправляет SMTP-команду DATA, переводящую соединение в режим передачи содержимого письма. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.
- data (VARCHAR2) – текст письма (включая заголовки и тело).

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Отправка готового письма в одном вызове.
- Завершение формирования письма в рамках SMTP-сеанса.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rcpt(v_conn, 'receiver@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_data(v_conn, 'Test email body content.'::varchar2);

    RAISE INFO 'Result utl_smtp_data: body sent in one call';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.OPEN_DATA

Процедура инициирует блок передачи данных письма после команды DATA, открывая поток для записи содержимого. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.

Выходные параметры:

- Возвращает дескриптор открытого потока для записи данных письма.

Типичные сценарии использования:

- Отправка больших сообщений частями.
- Последовательная запись заголовков и тела письма в поток.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rcpt(v_conn, 'receiver@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_open_data(v_conn);

    RAISE INFO 'Result utl_smtp_open_data: DATA mode opened';

    PERFORM utl_smtp_close_data(v_conn);
    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.WRITE_DATA

Процедура записывает часть данных письма в открытый поток, созданный вызовом OPEN_DATA. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.
- data (VARCHAR2) – строка с частью содержимого письма.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Поэтапная отправка больших писем или бинарных вложений.
- Управление структурой и форматированием письма на уровне потока.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rcpt(v_conn, 'receiver@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_open_data(v_conn);
    PERFORM utl_smtp_write_data(v_conn, 'Line 1 of message body.'::varchar2);

    RAISE INFO 'Result utl_smtp_write_data: Line 1 of message body.';

    PERFORM utl_smtp_close_data(v_conn);
    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.CLOSE_DATA

Процедура завершает передачу данных письма, закрывая поток, открытый функцией OPEN_DATA. Внимание! Данная процедура требует настройки параметра SMTP-сервера в файле postgresql.conf. Подробнее читайте в главе 4.5.2. Каталог базы данных.

Входящие параметры:

- c (PL/SQL RECORD) – активное SMTP-соединение.

Выходные параметры:

- Отсутствуют.

Типичные сценарии использования:

- Завершение записи тела письма и передача его серверу.
- Подтверждение конца данных в SMTP-сеансе перед отправкой команды QUIT.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rcpt(v_conn, 'receiver@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_open_data(v_conn);
    PERFORM utl_smtp_write_data(v_conn, 'Line before close.'::varchar2);
    PERFORM utl_smtp_close_data(v_conn);

    RAISE INFO 'Result utl_smtp_close_data: DATA mode closed';

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.VRFY

Функция проверяет адрес получателя с помощью SMTP-команды `VRFY`.

Входящие параметры:

- c (INTEGER) – дескриптор SMTP-соединения.
- recipient (VARCHAR2) – адрес или имя пользователя для проверки.

Выходные параметры:

- Возвращает строку ответа SMTP-сервера.

Типичные сценарии использования:

- Проверка существования получателя на SMTP-сервере.
- Диагностика доступности адресатов.

- Отладка SMTP-взаимодействия в тестовой среде.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
    v_res  varchar2;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    v_res := utl_smtp_vrfy(v_conn, 'testuser@example.com'::varchar2);

    RAISE INFO 'Result utl_smtp_vrfy: %', v_res;

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.WRITE_RAW_DATA

Процедура записывает бинарный RAW-фрагмент в тело SMTP-сообщения.

Входящие параметры:

- c (INTEGER) – дескриптор SMTP-соединения.
- data (RAW/BLOB) – бинарный фрагмент данных для передачи.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Передача бинарных фрагментов в теле письма.
- Формирование MIME-сообщений и вложений.

- Отправка данных без текстового преобразования.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
    v_raw  blob := '\x48656c6c6f20526177'::bytea;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    PERFORM utl_smtp_mail(v_conn, 'sender@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_rcpt(v_conn, 'receiver@example.com'::varchar2, NULL::varchar2);
    PERFORM utl_smtp_open_data(v_conn);
    PERFORM utl_smtp_write_raw_data(v_conn, v_raw);

    RAISE INFO 'Result utl_smtp_write_raw_data: raw fragment written';

    PERFORM utl_smtp_close_data(v_conn);
    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.COMMAND

Функция отправляет произвольную SMTP-команду и возвращает ответ сервера.

Входящие параметры:

- c (INTEGER) – дескриптор SMTP-соединения.
- command (VARCHAR2) – SMTP-команда.
- argument (VARCHAR2) – дополнительный аргумент команды.

Выходные параметры:

- Возвращает строку ответа SMTP-сервера.

Типичные сценарии использования:

- Выполнение нестандартных SMTP-команд.
- Отладка SMTP-сессий.
- Проверка реакции сервера на произвольные команды.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
    v_res  varchar2;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    v_res := utl_smtp_command(v_conn, 'NOOP'::varchar2, NULL::varchar2);

    RAISE INFO 'Result utl_smtp_command: %', v_res;

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.COMMAND_REPLIES

Функция отправляет SMTP-команду, ожидая многострочный ответ сервера.

Входящие параметры:

- c (INTEGER) – дескриптор SMTP-соединения.
- command (VARCHAR2) – SMTP-команда.
- argument (VARCHAR2) – дополнительный аргумент команды.

Выходные параметры:

- Возвращает текст многострочного ответа SMTP-сервера.

Типичные сценарии использования:

- Получение многострочных ответов SMTP-сервера.
- Работа с командами типа `HELP`.
- Отладка расширенных возможностей SMTP-сервера.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
    v_res  varchar2;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);
    v_res := utl_smtp_command_replies(v_conn, 'HELP'::varchar2, NULL::varchar2);

    RAISE INFO 'Result utl_smtp_command_replies: %', v_res;

    PERFORM utl_smtp_quit(v_conn);
END $$;
```

UTL_SMTP.STARTTLS

Функция инициирует SMTP-команду `STARTTLS`.

Входящие параметры:

- c (INTEGER) – дескриптор SMTP-соединения.
- wallet_path (VARCHAR2) – путь к хранилищу сертификатов или дополнительный параметр TLS.

Выходные параметры:

- Возвращает строку ответа SMTP-сервера или сообщение об ошибке.

Типичные сценарии использования:

- Перевод SMTP-соединения в защищённый режим.
- Проверка поддержки TLS на сервере.
- Диагностика настройки защищённой почтовой доставки.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
    v_res  varchar2;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);

    BEGIN
        v_res := utl_smtp_starttls(v_conn, NULL::varchar2);
        RAISE INFO 'Result utl_smtp_starttls: %', v_res;
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'Result utl_smtp_starttls: %', SQLERRM;
    END;

    PERFORM utl_smtp_close_connection(v_conn);
END $$;
```

UTL_SMTP.HELP

Функция возвращает справочную информацию SMTP-сервера по указанной команде.

Входящие параметры:

- c (INTEGER) – дескриптор SMTP-соединения.
- command (VARCHAR2) – команда, для которой запрашивается справка.

Выходные параметры:

- Возвращает строку ответа SMTP-сервера или сообщение об ошибке.

Типичные сценарии использования:

- Получение справки по SMTP-командам.
- Проверка набора поддерживаемых команд сервера.
- Диагностика SMTP-сервера в тестовой среде.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
    v_res  varchar2;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);

    BEGIN
        v_res := utl_smtp_help(v_conn, 'HELP'::varchar2);
        RAISE INFO 'Result utl_smtp_help: %', v_res;
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'Result utl_smtp_help: %', SQLERRM;
    END;

    PERFORM utl_smtp_close_connection(v_conn);
END $$;
```

UTL_SMTP.AUTH

Функция выполняет SMTP-аутентификацию с использованием указанных учётных данных.

Входящие параметры:

- c (INTEGER) – дескриптор SMTP-соединения.
- username (VARCHAR2) – имя пользователя.
- password (VARCHAR2) – пароль.
- scheme (VARCHAR2) – схема аутентификации.

Выходные параметры:

- Возвращает строку ответа SMTP-сервера или сообщение об ошибке.

Типичные сценарии использования:

- Проверка SMTP-аутентификации в тестовой среде.
- Настройка авторизованной отправки почты.
- Диагностика поддерживаемых схем входа на почтовом сервере.

Примеры запросов:

```
DO $$
DECLARE
    v_conn integer;
    v_res  varchar2;
BEGIN
    v_conn := utl_smtp_open_connection(
        '127.0.0.1'::varchar2,
        2525,
        10,
        NULL::varchar2,
        NULL::varchar2,
        false,
        NULL::varchar2
    );

    PERFORM utl_smtp_ehlo(v_conn, 'example.com'::varchar2);

    BEGIN
        v_res := utl_smtp_auth(
            v_conn,
            'test_user'::varchar2,
            'test_pass'::varchar2,
            'PLAIN'::varchar2
        );
        RAISE INFO 'Result utl_smtp_auth: %', v_res;
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'Result utl_smtp_auth: %', SQLERRM;
    END;

    PERFORM utl_smtp_close_connection(v_conn);
END $$;
```

UTL_SMTP_DATA_EXT

Вспомогательная функция для: utl_smtp_data

Описание: отправляет тело сообщения через SMTP-соединение.

Входящие параметры:

- connection (INT4) – идентификатор соединения
- data (VARCHAR2) – данные для отправки
- line_mode (INT4) – режим обработки строк

Выходные параметры:

- VOID – функция не возвращает значения

Типичные сценарии использования:

- Отправка тела email-сообщения
- SMTP-коммуникация

Примеры запросов:

```
PERFORM utl_smtp_data_ext(conn, 'Hello', 3);
```

UTL_SMTP_GET_VERSION

Функция возвращает номер версии пакета UTL_SMTP для работы с SMTP-протоколом.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для отправки почты
- Совместимость команд SMTP

Примеры запросов:

```
SELECT utl_smtp_get_version();
```

5.2.42. Пакет UTL_TCP

UTL_TCP.OPEN_CONNECTION

Функция открывает TCP-соединение с удаленным сервером и возвращает дескриптор соединения для использования в других функциях пакета.

Входящие параметры:

- remote_host (VARCHAR2) – имя хоста или IP-адрес удаленного сервера.
- remote_port (PLS_INTEGER) – номер порта на удаленном сервере.
- local_host (VARCHAR2) – имя хоста или IP-адрес локального интерфейса для исходящего соединения (опционально).
- local_port (PLS_INTEGER) – номер локального порта (опционально, обычно 0).
- charset (VARCHAR2) – кодировка символов для текстового обмена (по умолчанию берется из сессии).
- newline (CHAR) – символ(ы) конца строки (CRLF по умолчанию).
- tx_timeout (PLS_INTEGER) – таймаут отправки данных в секундах.
- rx_timeout (PLS_INTEGER) – таймаут получения данных в секундах.

Выходные параметры:

- connection (PLS_INTEGER) – дескриптор открытого соединения.

Типичные сценарии использования:

- Взаимодействие с веб-сервисами (HTTP/HTTPS) по протоколам TCP.
- Обмен данными с сокетами других приложений (Java, .NET, Python).
- Интеграция со сторонними системами, предоставляющими TCP-интерфейсы.
- Отправка данных на удаленные серверы мониторинга или логирования.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    PERFORM utl_tcp_close_connection(conn);
    RAISE NOTICE 'OK';
END $$;
```

UTL_TCP.CLOSE_CONNECTION

Процедура закрывает указанное TCP-соединение.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения, возвращенный OPEN_CONNECTION.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Корректное завершение сетевого соединения после завершения обмена данными.
- Освобождение сетевых ресурсов на стороне сервера и клиента.
- Обработка ошибок для гарантированного закрытия соединения.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    PERFORM utl_tcp_close_connection(conn);
    RAISE NOTICE 'OK';
END $$;
```

UTL_TCP.WRITE_TEXT

Процедура отправляет текстовые данные (VARCHAR2) через открытое TCP-соединение.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.
- data (VARCHAR2) – текст для отправки.
- len (PLS_INTEGER) – количество символов для отправки. Если NULL, отправляется вся строка.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Отправка текстовых запросов к HTTP-серверу или другим текстовым протоколам (SMTP, FTP-команды).
- Передача структурированных данных в формате XML или JSON.
- Отправка логов или уведомлений на удаленный сервер.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Отправлено байт: %', utl_tcp_write_text(conn, 'Привет, мир!', NULL);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.GET_TEXT

Функция читает текстовые данные из TCP-соединения.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.
- len (PLS_INTEGER) – максимальное количество символов для чтения. Если NULL, читаются все доступные данные до таймаута.

Выходные параметры:

- Возвращает прочитанные текстовые данные.

Типичные сценарии использования:

- Получение текстовых ответов от веб-сервера (HTML, JSON, XML).
- Чтение строковых данных из сокета другого приложения.
- Получение статусных сообщений от удаленного сервиса.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_get_text(conn, 13,
FALSE);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.FLUSH

Процедура принудительно отправляет все буферизированные данные в сокет.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Гарантированная отправка данных перед ожиданием ответа от сервера.
- Оптимизация производительности при отправке множества мелких сообщений (отправка одним пакетом).
- Работа в режиме без буферизации (если требуется немедленная отправка).

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    PERFORM utl_tcp_flush(conn); -- по сути ничего не делает, т.к. нет своего буфера
записи
    PERFORM utl_tcp_close_connection(conn);
    RAISE NOTICE 'OK';
END $$;
```

/

UTL_TCP.AVAILABLE

Функция возвращает количество байт, доступных для чтения из сокета без блокировки.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.

Выходные параметры:

- PLS_INTEGER – количество доступных байт.

Типичные сценарии использования:

- Проверка наличия данных перед попыткой чтения, чтобы избежать блокировки.
- Реализация неблокирующего чтения или цикла опроса (polling).
- Определение размера ожидаемого ответа от сервера.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE NOTICE 'Количество доступных байт: %', utl_tcp_available(conn, 0);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.WRITE_LINE

Процедура отправляет текстовую строку через TCP-соединение и добавляет в конец символ(ы) конца строки (newline).

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.
- data (VARCHAR2) – текстовая строка для отправки (без символов конца

строки).

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Отправка команд в интерактивные протоколы, где каждая команда заканчивается переводом строки (TELNET, SMTP, FTP).
- Передача логов в формате, где каждая запись – отдельная строка.
- Упрощение кода, когда требуется отправлять строки с завершающим newline.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Отправлено байт: %', utl_tcp_write_line(conn, 'Привет, мир!');
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

— utl_tcp_write_raw DO DECLARE conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL, NULL, 'UTF8', NULL, 10, NULL, NULL); BEGIN RAISE INFO 'Отправлено байт: %', utl_tcp_write_raw(conn, convert_to('Привет, мир!', 'UTF8')::RAW, NULL); PERFORM utl_tcp_close_connection(conn); END ;

UTL_TCP.GET_LINE

Функция читает одну текстовую строку из TCP-соединения. Чтение останавливается при получении символов конца строки (newline) или достижении таймаута.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.
- remove_crlf (BOOLEAN) – флаг, указывающий, нужно ли удалять символы конца строки из результата (по умолчанию TRUE).

Выходные параметры:

- Возвращает прочитанную строку (без символов конца строки, если remove_crlf=TRUE).

Типичные сценарии использования:

- Чтение ответов от сервера построчно (например, чтение заголовков HTTP или ответов SMTP).
- Интерактивное взаимодействие с сокет-сервером, который отправляет данные построчно.
- Парсинг текстовых протоколов, где данные разделены переводами строк.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_get_line(conn, TRUE,
FALSE);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

— utl_tcp_get_line_nchar DO DECLARE conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL, NULL, 'UTF8', NULL, 10, NULL, NULL); BEGIN RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_get_line_nchar(conn, TRUE, FALSE); PERFORM utl_tcp_close_connection(conn); END ;

UTL_TCP.WRITE_RAW

Процедура отправляет двоичные данные (RAW) через открытое TCP-соединение.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.
- data (RAW) – двоичные данные для отправки.
- len (PLS_INTEGER) – количество байт для отправки. Если NULL, отправляются все данные.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Отправка файлов, изображений или любых бинарных данных через сокет.
- Передача зашифрованных или сжатых данных.
- Реализация пользовательских двоичных протоколов поверх TCP.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Отправлено байт: %', utl_tcp_write_raw(conn, convert_to('Привет,
мир!', 'UTF8')::RAW, NULL);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.GET_RAW

Функция читает двоичные данные (RAW) из TCP-соединения.

Входящие параметры:

- c (PLS_INTEGER) – дескриптор соединения.
- len (PLS_INTEGER) – максимальное количество байт для чтения. Если NULL, читаются все доступные данные до таймаута.

Выходные параметры:

RAW – прочитанные двоичные данные.

Типичные сценарии использования:

- Получение бинарных файлов, загружаемых по протоколам (HTTP, FTP).
- Чтение двоичных ответов от сервисов (например, получение картинки или PDF).
- Работа с протоколами, которые передают данные в смешанном формате (текст + двоичные вложения).

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_get_raw(conn, 12,
FALSE);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.CLOSE_ALL_CONNECTIONS

Процедура принудительно закрывает все открытые TCP-соединения, созданные в текущей сессии.

Входящие параметры:

Параметры отсутствуют.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Аварийная очистка сетевых ресурсов при возникновении критической ошибки в приложении.
- Гарантированное освобождение соединений перед завершением сессии или процедуры.
- Использование в блоках исключений для предотвращения утечек соединений.

Примеры запросов:

```
DO $$
BEGIN
    PERFORM utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,  NULL,
'UTF8', NULL,  10, NULL,  NULL);
    PERFORM utl_tcp_open_connection('localhost', 9001, NULL, NULL, NULL,  NULL,
'UTF8', NULL,  10, NULL,  NULL);
    PERFORM utl_tcp_close_all_connections();
    RAISE NOTICE 'OK';
END $$;
```

UTL_TCP.GET_TEXT_NCHAR

Функция считывает из TCP-соединения текст в национальной кодировке.

Входящие параметры:

- c (INTEGER) – дескриптор TCP-соединения.
- len (INTEGER) – количество символов для чтения.
- peek (BOOLEAN) – признак чтения без удаления данных из входного буфера.

Выходные параметры:

- Возвращает считанный текст в национальной кодировке.

Типичные сценарии использования:

- Чтение символьных данных из TCP-сервера.
- Работа с текстом в заданной кодировке соединения.
- Тестирование низкоуровневого сетевого обмена.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_get_text_nchar(conn,
13, FALSE);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.READ_LINE

Функция считывает одну строку из TCP-соединения.

Входящие параметры:

- c (INTEGER) – дескриптор TCP-соединения.
- remove_crlf (BOOLEAN) – признак удаления символов конца строки.

Выходные параметры:

- Возвращает считанную строку.

Типичные сценарии использования:

- Построчное чтение данных из TCP-соединения.
- Обработка текстовых протоколов поверх TCP.
- Тестирование строкового обмена с TCP-сервером.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_read_line(conn,
FALSE);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.READ_RAW

Функция считывает бинарные данные из TCP-соединения.

Входящие параметры:

- c (INTEGER) – дескриптор TCP-соединения.
- len (INTEGER) – количество байт для чтения.
- peek (BOOLEAN) – признак чтения без удаления данных из входного буфера.

Выходные параметры:

- Возвращает считанные бинарные данные.

Типичные сценарии использования:

- Чтение бинарного содержимого из TCP-потока.
- Работа с протоколами, передающими двоичные данные.
- Тестирование низкоуровневого сетевого обмена без текстового преобразования.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_read_raw(conn, 13,
FALSE);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.READ_TEXT

Функция считывает текстовые данные из TCP-соединения.

Входящие параметры:

- c (INTEGER) – дескриптор TCP-соединения.
- len (INTEGER) – количество символов для чтения.
- peek (BOOLEAN) – признак чтения без удаления данных из входного буфера.

Выходные параметры:

- Возвращает считанный текст.

Типичные сценарии использования:

- Чтение текстовых сообщений из TCP-потока.
- Обработка прикладных текстовых протоколов.
- Тестирование чтения данных с TCP-сервера.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    RAISE INFO 'Полученные данные от TCP сервера: "%"', utl_tcp_read_text(conn, 13,
FALSE);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP.SECURE_CONNECTION

Процедура переводит TCP-соединение в защищённый режим.

Входящие параметры:

- c (INTEGER) – дескриптор TCP-соединения.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Установка защищённого соединения поверх TCP.
- Подготовка канала к передаче чувствительных данных.
- Проверка поддержки защищённого режима в тестовой среде.

Примеры запросов:

```
DO $$
DECLARE
    conn integer := utl_tcp_open_connection('localhost', 9000, NULL, NULL, NULL,
NULL, 'UTF8', NULL,  10, NULL,  NULL);
BEGIN
    PERFORM utl_tcp_secure_connection(conn);
    PERFORM utl_tcp_close_connection(conn);
END $$;
```

UTL_TCP_BYTES_AVAILABLE_FD

Вспомогательная функция для: utl_tcp_available

Описание: возвращает количество доступных байт в сокете.

Входящие параметры:

- fd (INT4) – файловый дескриптор сокета

Выходные параметры:

- INT4 – количество доступных байт

Типичные сценарии использования:

- Проверка наличия данных перед чтением
- Асинхронная работа с сокетами

Примеры запросов:

```
SELECT utl_tcp_bytes_available_fd(1559);
```

UTL_TCP_GET_VERSION

Функция возвращает номер версии пакета UTL_TCP для работы с TCP/IP-соединениями.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для сетевых соединений
- Совместимость сокетов

Примеры запросов:

```
SELECT utl_tcp_get_version();
```

5.2.43. Пакет UTL_URL

UTL_URL.ESCAPE

Функция выполняет экранирование специальных символов в URL в соответствии со стандартом RFC 3986. Заменяет небезопасные символы на их процентное представление.

Входящие параметры:

- url (VARCHAR2) – исходная строка URL для экранирования.
- escape_reserved_chars (BOOLEAN) – флаг, указывающий экранироваybt зарезервированных символов (TRUE - экранировать, FALSE - не экранировать).

Выходные параметры:

- Возвращает строку с экранированным URL.

Типичные сценарии использования:

- Подготовка URL для HTTP-запросов.
- Экранирование параметров в веб-ссылках.
- Обработка пользовательского ввода перед использованием в URL.
- Создание корректных веб-адресов с пробелами и специальными символами.

Примеры запросов:

```
SELECT UTL_URL_ESCAPE('http://www.acme.com/a url with space.html', FALSE, 'UTF-8');
```

UTL_URL.UNESCAPE

Функция выполняет декодирование экранированных символов в URL, преобразуя процентное представление обратно в оригинальные символы в соответствии со стандартом RFC 3986.

Входящие параметры:

- url (VARCHAR2) – строка с экранированным URL для декодирования.
- charset (VARCHAR2) – кодировка символов для декодирования (по умолчанию

'UTF-8').

Выходные параметры:

- Возвращает строку с декодированным URL.

Типичные сценарии использования:

- Обработка полученных URL-параметров в веб-приложениях.
- Декодирование данных из HTTP-запросов.
- Чтение и обработка экранированных путей и ссылок.
- Восстановление оригинального текста из процентного представления.

Примеры запросов:

```
select utl_url_unescape('abc%20def', 'UTF-8');
```

UTL_URL_IS_RESERVED

Вспомогательная функция для: utl_url_escape

Описание: проверяет, является ли символ зарезервированным в URL.

Входящие параметры:

- c (INT2) – код символа

Выходные параметры:

- BOOLEAN – TRUE если символ зарезервирован

Типичные сценарии использования:

- Экранирование URL-символов
- Формирование корректных URL

Примеры запросов:

```
SELECT utl_url_is_reserved(ASCII(';')::smallint);
```

UTL_URL_GET_VERSION

Функция возвращает номер версии пакета UTL_URL для кодирования и декодирования URL.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для URL-функций
- Совместимость кодирования

Примеры запросов:

```
SELECT utl_url_get_version();
```

5.2.44. Пакет UTL_CALL_STACK

UTL_CALL_STACK.OWNER

Функция возвращает имя владельца объекта (схемы), в котором находится единица программы (юнит) на указанной глубине стека вызовов.

Входящие параметры:

- depth (PLS_INTEGER) – глубина в стеке вызовов (по умолчанию 0 – текущая единица).

Выходные параметры:

- Возвращает имя владельца схемы.

Типичные сценарии использования:

- Отладка сложных цепочек вызовов процедур и функций.
- Аудит и логирование для определения источника вызова.
- Динамическое принятие решений в зависимости от контекста выполнения (например, проверка прав).

Примеры запросов:

```
select utl_call_stack_owner(1);
```

UTL_CALL_STACK.CURRENT_EDITION

Функция возвращает имя текущей (активной) редакции базы данных, в контексте которой выполняется код.

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- Возвращает имя текущей редакции (edition).

Типичные сценарии использования:

- Проверка контекста выполнения в средах, использующих редакции для обновления приложений (Edition-Based Redefinition).
- Логирование и отладка в многоконных средах.

Примеры запросов:

```
select utl_call_stack_current_edition(1);
```

UTL_CALL_STACK.BACKTRACE_DEPTH

Функция возвращает количество доступных записей в трассировке стека (backtrace). Backtrace содержит последовательность вызовов, которая привела к текущей точке выполнения, включая вызовы, не связанные с PL/SQL.

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- Возвращает глубину трассировки стека (backtrace depth).

Типичные сценарии использования:

- Получение полной картины стека вызовов, включая вызовы извне PL/SQL.
- Создание подробных логов ошибок с полной трассировкой.
- Диагностика сложных взаимодействий между компонентами.

Примеры запросов:

```
select utl_call_stack_backtrace_depth();
```

UTL_CALL_STACK.ERROR_DEPTH

Функция возвращает количество записей в стеке вызовов на момент возникновения последнего исключения. Эта глубина соответствует контексту, в котором было поднято исключение.

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- Возвращает глубину стека в точке ошибки.

Типичные сценарии использования:

- Анализ стека вызовов внутри блока исключений (WHEN OTHERS) для определения источника ошибки.
- Построение пользовательских сообщений об ошибках с указанием места возникновения.

Примеры запросов:

```
select utl_call_stack_error_depth();
```

UTL_CALL_STACK.DYNAMIC_DEPTH

Функция возвращает текущую глубину динамического стека вызовов. Динамический стек включает вызовы, выполненные через динамический SQL (например, EXECUTE IMMEDIATE).

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- Возвращает глубину динамического стека.

Типичные сценарии использования:

- Отладка кода, использующего динамический SQL.
- Понимание вложенности вызовов, генерируемых во время выполнения.

Примеры запросов:

```
select utl_call_stack_dynamic_depth();
```

UTL_CALL_STACK.LEXICAL_DEPTH

Функция возвращает лексическую глубину (уровень вложенности) подпрограммы (процедуры или функции) в указанной единице программы на заданной глубине стека.

Входящие параметры:

- depth (PLS_INTEGER) – глубина в стеке вызовов (по умолчанию 0 – текущая единица).

Выходные параметры:

- Возвращает лексическую глубину (0 для внешней подпрограммы, 1 для вложенной и т.д.).

Типичные сценарии использования:

- Определение контекста вложенных подпрограмм.
- Проверка структуры кода при рефакторинге или анализе.

Примеры запросов:

```
select utl_call_stack_lexical_depth(1);
```

*UTL_CALL_STACK.*ERROR_NUMBER

Функция возвращает номер ошибки последнего исключения, произошедшего в сеансе.

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- Возвращает номер ошибки (например, -6502 для VALUE_ERROR).

Типичные сценарии использования:

- Получение кода ошибки в блоке WHEN OTHERS для последующей обработки или логирования.
- Интеграция с системами мониторинга и оповещения.

Примеры запросов:

```
select utl_call_stack_error_number(1);
```

UTL_CALL_STACK.BACKTRACE_UNIT

Функция возвращает имя единицы программы (юнита) на указанной глубине трассировки стека (backtrace).

Входящие параметры:

- backtrace_depth (PLS_INTEGER) – глубина в трассировке стека.

Выходные параметры:

- Возвращает имя единицы программы (например, имя пакета, процедуры, функции).

Типичные сценарии использования:

- Построение детализированной трассировки стека для логирования или отображения пользователю.
- Анализ последовательности вызовов, приведших к ошибке.

Примеры запросов:

```
select utl_call_stack_backtrace_unit(1);
```

UTL_CALL_STACK.ERROR_MSG

Функция возвращает текстовое сообщение последнего исключения, произошедшего в сеансе.

Входящие параметры:

- Параметры отсутствуют.

Выходные параметры:

- Возвращает текст сообщения об ошибке.

Типичные сценарии использования:

- Логирование понятных сообщений об ошибках.
- Формирование пользовательских уведомлений с деталями исходной ошибки.

Примеры запросов:

```
select utl_call_stack_error_msg(1);
```

UTL_CALL_STACK_COMPOSE_SUBPROGRAM

Вспомогательная функция для: utl_call_stack_subprogram, utl_call_stack_concatenate_subprogram

Описание: формирует строковое представление подпрограммы в стеке вызовов.

Входящие параметры:

- name (VARCHAR2) – имя подпрограммы
- format_mask (INT4) – маска форматирования

Выходные параметры:

- VARCHAR2 – отформатированное представление подпрограммы

Типичные сценарии использования:

- Отладка и трассировка вызовов
- Формирование информации о стеке

Примеры запросов:

```
SELECT utl_call_stack_compose_subprogram('proc',1);
```

UTL_CALL_STACK_GET_VERSION

Функция возвращает номер версии пакета UTL_CALL_STACK для получения информации о стеке вызовов.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для отладки
- Совместимость функций стека

Примеры запросов:

```
SELECT utl_call_stack_get_version();
```

5.2.45. Пакет UTL_HTTP

UTL_HTTP.SET_TRANSFER_TIMEOUT

Процедура задаёт таймаут передачи данных для HTTP-запросов текущей сессии.

Входящие параметры:

- timeout (INTEGER) – таймаут в секундах.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Настройка максимального времени ожидания при работе с HTTP.
- Подготовка окружения перед вызовами UTL_HTTP.
- Ограничение времени выполнения сетевых операций.

Примеры запросов:

```
DO $$
DECLARE
    timeout_val INTEGER;
BEGIN
    -- Подготовка данных (не требуется)

    -- Тест
    PERFORM UTL_HTTP_SET_TRANSFER_TIMEOUT(10);
    timeout_val := UTL_HTTP_GET_TRANSFER_TIMEOUT();

    IF timeout_val != 10 THEN
        RAISE EXCEPTION 'UTL_HTTP_SET_TRANSFER_TIMEOUT failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_SET_TRANSFER_TIMEOUT: OK';

    -- Очистка
END;
$$;
```

UTL_HTTP.GET_TRANSFER_TIMEOUT

Функция возвращает текущий таймаут передачи данных для HTTP-запросов.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает таймаут передачи данных в секундах.

Типичные сценарии использования:

- Проверка текущих настроек HTTP-сессии.

- Контроль значений после изменения конфигурации.
- Отладка сетевых параметров пакета.

Примеры запросов:

```
DO $$
DECLARE
    timeout_val INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_TRANSFER_TIMEOUT(25);

    -- Тест
    timeout_val := UTL_HTTP_GET_TRANSFER_TIMEOUT();

    IF timeout_val != 25 THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_TRANSFER_TIMEOUT failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_TRANSFER_TIMEOUT: OK';

    -- Очистка
END;
$$;
```

UTL_HTTP.SET_RESPONSE_ERROR_CHECK

Процедура включает или отключает проверку HTTP-ответов на ошибки.

Входящие параметры:

- enable (BOOLEAN) – признак включения проверки ошибок HTTP-ответа.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Управление режимом обработки ошибочных HTTP-ответов.
- Настройка поведения при получении кодов ошибок сервера.
- Подготовка сценариев с контролем исключений.

Примеры запросов:

```
DO $$
BEGIN
    -- Тест
    PERFORM UTL_HTTP_SET_RESPONSE_ERROR_CHECK(TRUE);

    IF UTL_HTTP_GET_RESPONSE_ERROR_CHECK() != TRUE THEN
        RAISE EXCEPTION 'UTL_HTTP_SET_RESPONSE_ERROR_CHECK failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_SET_RESPONSE_ERROR_CHECK: OK';
END;
$$;
```

UTL_HTTP.GET_RESPONSE_ERROR_CHECK

Функция возвращает признак включённой проверки HTTP-ответов на ошибки.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает TRUE, если проверка включена, иначе FALSE.

Типичные сценарии использования:

- Проверка текущего режима обработки HTTP-ошибок.
- Контроль конфигурации перед выполнением запросов.
- Диагностика поведения HTTP-клиента.

Примеры запросов:

```
DO $$
DECLARE
    err_check BOOLEAN;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_RESPONSE_ERROR_CHECK(FALSE);

    -- Тест
    err_check := UTL_HTTP_GET_RESPONSE_ERROR_CHECK();

    IF err_check != FALSE THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_RESPONSE_ERROR_CHECK failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_RESPONSE_ERROR_CHECK: OK';
END;
$$;
```

UTL_HTTP.SET_FOLLOW_REDIRECT_SESSION

Процедура задаёт максимальное количество перенаправлений для HTTP-запросов текущей сессии.

Входящие параметры:

- max_redirects (INTEGER) – максимально допустимое число перенаправлений.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Настройка поведения клиента при HTTP-редиректах.
- Ограничение числа автоматических переходов.
- Подготовка окружения перед сетевыми запросами.

Примеры запросов:

```
DO $$
BEGIN
    -- Тест
    PERFORM UTL_HTTP_SET_FOLLOW_REDIRECT_SESSION(7);

    IF UTL_HTTP_GET_FOLLOW_REDIRECT() != 7 THEN
        RAISE EXCEPTION 'UTL_HTTP_SET_FOLLOW_REDIRECT_SESSION failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_SET_FOLLOW_REDIRECT_SESSION: OK';
END;
$$;
```

UTL_HTTP.GET_FOLLOW_REDIRECT

Функция возвращает текущее значение лимита перенаправлений HTTP-сессии.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает количество разрешённых перенаправлений.

Типичные сценарии использования:

- Проверка текущей конфигурации обработки редиректов.
- Отладка HTTP-настроек сессии.
- Контроль изменений после настройки параметров.

Примеры запросов:

```
DO $$
DECLARE
    redirect_val INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_FOLLOW_REDIRECT_SESSION(3);

    -- Тест
    redirect_val := UTL_HTTP_GET_FOLLOW_REDIRECT();

    IF redirect_val != 3 THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_FOLLOW_REDIRECT failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_FOLLOW_REDIRECT: OK';
END;
$$;
```

UTL_HTTP.SET_BODY_CHARSET_SESSION

Процедура задаёт кодировку тела HTTP-сообщения для текущей сессии.

Входящие параметры:

- charset (VARCHAR2) – имя кодировки тела HTTP-сообщения.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Настройка кодировки текста при HTTP-обмене.
- Подготовка сессии для работы с локализованными данными.
- Управление интерпретацией символьного содержимого.

Примеры запросов:

```
DO $$
BEGIN
    -- Тест
    PERFORM UTL_HTTP_SET_BODY_CHARSET_SESSION('UTF-8');

    IF UTL_HTTP_GET_BODY_CHARSET()::TEXT != 'UTF-8' THEN
        RAISE EXCEPTION 'UTL_HTTP_SET_BODY_CHARSET_SESSION failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_SET_BODY_CHARSET_SESSION: OK';
END;
$$;
```

UTL_HTTP.GET_BODY_CHARSET

Функция возвращает текущую кодировку тела HTTP-сообщения.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает имя текущей кодировки тела сообщения.

Типичные сценарии использования:

- Проверка активной кодировки HTTP-сессии.
- Контроль параметров после их изменения.
- Отладка обработки текстовых HTTP-данных.

Примеры запросов:

```
DO $$
DECLARE
    charset_val VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_BODY_CHARSET_SESSION('windows-1251');

    -- Тест
    charset_val := UTL_HTTP_GET_BODY_CHARSET();

    IF charset_val::TEXT != 'windows-1251' THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_BODY_CHARSET failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_BODY_CHARSET: OK';
END;
$$;
```

UTL_HTTP.SET_PROXY

Процедура задаёт параметры HTTP-прокси для текущей сессии.

Входящие параметры:

- proxy (VARCHAR2) – адрес прокси-сервера.
- no_proxy_domains (VARCHAR2) – список доменов, для которых прокси не используется.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Настройка выхода в сеть через прокси-сервер.
- Исключение локальных адресов из проксирования.
- Подготовка HTTP-запросов в корпоративной сети.

Примеры запросов:

```
DO $$
DECLARE
    proxy_out VARCHAR2;
    no_proxy_out VARCHAR2;
BEGIN
    -- Тест
    PERFORM UTL_HTTP_SET_PROXY('http://proxy.test:8080', 'localhost,127.0.0.1');

    SELECT proxy, no_proxy_domains INTO proxy_out, no_proxy_out FROM
UTL_HTTP_GET_PROXY();

    IF proxy_out::TEXT != 'http://proxy.test:8080' THEN
        RAISE EXCEPTION 'UTL_HTTP_SET_PROXY failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_SET_PROXY: OK';
END;
$$;
```

UTL_HTTP.GET_PROXY

Функция возвращает текущие настройки HTTP-прокси.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает сведения о прокси и списке исключений.

Типичные сценарии использования:

- Проверка текущей конфигурации прокси.
- Диагностика сетевых настроек HTTP-клиента.
- Контроль параметров после их установки.

Примеры запросов:

```
DO $$
DECLARE
    proxy_out VARCHAR2;
    no_proxy_out VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY('http://proxy.example.com:3128', '*.local,10.0.0.*');

    -- Тест
    SELECT proxy, no_proxy_domains INTO proxy_out, no_proxy_out FROM
UTL_HTTP_GET_PROXY();

    IF proxy_out::TEXT != 'http://proxy.example.com:3128' THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_PROXY failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_PROXY: OK';
END;
$$;
```

UTL_HTTP.CREATE_REQUEST_CONTEXT

Функция создаёт контекст HTTP-запроса для последующего использования.

Входящие параметры:

- wallet_path (VARCHAR2) – путь к хранилищу сертификатов.
- wallet_password (VARCHAR2) – пароль хранилища.
- enable_cookies (BOOLEAN) – признак использования cookie.
- max_cookies (INTEGER) – максимальное число cookie.
- max_cookies_per_site (INTEGER) – максимальное число cookie на сайт.

Выходные параметры:

- Возвращает идентификатор контекста HTTP-запроса.

Типичные сценарии использования:

- Создание отдельного контекста выполнения HTTP-запроса.
- Изоляция параметров запросов между сценариями.
- Подготовка окружения для сложных HTTP-вызовов.

Примеры запросов:

```
DO $$
DECLARE
    v_ctx INTEGER;
BEGIN
    -- Тест
    v_ctx := UTL_HTTP_CREATE_REQUEST_CONTEXT(NULL::VARCHAR2, NULL::VARCHAR2, FALSE, 0,
0);

    IF v_ctx IS NULL OR v_ctx <= 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_CREATE_REQUEST_CONTEXT failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_CREATE_REQUEST_CONTEXT: OK';

    -- Очистка
    PERFORM UTL_HTTP_DESTROY_REQUEST_CONTEXT(v_ctx);
END;
$$;
```

UTL_HTTP.DESTROY_REQUEST_CONTEXT

Процедура уничтожает ранее созданный контекст HTTP-запроса.

Входящие параметры:

- ctx (INTEGER) – идентификатор контекста HTTP-запроса.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Освобождение ресурсов после завершения HTTP-сценария.
- Корректное завершение работы с контекстом запроса.
- Очистка временных сетевых настроек.

Примеры запросов:

```
DO $$
DECLARE
    v_ctx INTEGER;
BEGIN
    -- Подготовка
    v_ctx := UTL_HTTP_CREATE_REQUEST_CONTEXT(NULL::VARCHAR2, NULL::VARCHAR2, FALSE, 0,
0);

    -- Тест
    PERFORM UTL_HTTP_DESTROY_REQUEST_CONTEXT(v_ctx);

    RAISE NOTICE 'UTL_HTTP_DESTROY_REQUEST_CONTEXT: OK';
END;
$$;
```

UTL_HTTP.BEGIN_REQUEST

Функция создаёт HTTP-запрос и возвращает его дескриптор.

Входящие параметры:

- url (VARCHAR2) – адрес HTTP-ресурса.
- method (VARCHAR2) – HTTP-метод запроса.
- http_version (VARCHAR2) – версия HTTP-протокола.
- request_context (INTEGER) – идентификатор контекста запроса.
- https_host (VARCHAR2) – имя HTTPS-хоста.

Выходные параметры:

- Возвращает дескриптор HTTP-запроса.

Типичные сценарии использования:

- Инициализация HTTP-запроса перед отправкой.
- Подготовка дескриптора для установки заголовков и тела.
- Начало многошагового HTTP-сценария.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);

    -- Тест
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);

    IF req_h IS NULL OR req_h <= 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_BEGIN_REQUEST failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_BEGIN_REQUEST: OK';
END;
$$;
```

UTL_HTTP.SET_HEADER

Процедура добавляет или изменяет HTTP-заголовок запроса.

Входящие параметры:

- req (INTEGER) – дескриптор HTTP-запроса.
- name (VARCHAR2) – имя заголовка.
- value (VARCHAR2) – значение заголовка.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Добавление служебных HTTP-заголовков.
- Указание типа содержимого или идентификатора клиента.
- Подготовка запроса перед отправкой.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);

    -- Тест
    PERFORM UTL_HTTP_SET_HEADER(req_h, 'User-Agent', 'RUDB-Tester/1.0');

    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    RAISE NOTICE 'UTL_HTTP_SET_HEADER: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.GET_RESPONSE

Функция отправляет HTTP-запрос и возвращает дескриптор ответа.

Входящие параметры:

- req (INTEGER) – дескриптор HTTP-запроса.
- return_info_response (BOOLEAN) – признак возврата информационного ответа.

Выходные параметры:

- Возвращает дескриптор HTTP-ответа.

Типичные сценарии использования:

- Получение HTTP-ответа после подготовки запроса.
- Переход к чтению заголовков и тела ответа.
- Организация клиентского взаимодействия с HTTP-сервером.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);

    -- Тест
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    IF resp_h IS NULL OR resp_h <= 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_RESPONSE failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_RESPONSE: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.GET_HEADER_COUNT

Функция возвращает количество заголовков в HTTP-ответе.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.

Выходные параметры:

- Возвращает число заголовков ответа.

Типичные сценарии использования:

- Анализ состава HTTP-ответа.
- Перебор заголовков ответа.
- Отладка сетевого взаимодействия.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    v_count INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    v_count := UTL_HTTP_GET_HEADER_COUNT(resp_h);

    IF v_count IS NULL OR v_count <= 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_HEADER_COUNT failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_HEADER_COUNT: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.GET_HEADER

Функция возвращает заголовок HTTP-ответа по его порядковому номеру.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.
- index (INTEGER) – порядковый номер заголовка.

Выходные параметры:

- Возвращает имя и значение выбранного заголовка.

Типичные сценарии использования:

- Получение отдельных заголовков ответа.
- Перебор заголовков по индексу.

- Анализ метаданных HTTP-ответа.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    v_name VARCHAR2;
    v_val VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    SELECT name, value INTO v_name, v_val FROM UTL_HTTP_GET_HEADER(resp_h, 1);

    IF v_name IS NULL OR v_name::TEXT = '' THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_HEADER failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_HEADER: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.GET_HEADER_BY_NAME

Функция возвращает значение HTTP-заголовка по его имени.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.
- name (VARCHAR2) – имя заголовка.
- instance (INTEGER) – номер вхождения заголовка.

Выходные параметры:

- Возвращает значение заголовка.

Типичные сценарии использования:

- Извлечение конкретного HTTP-заголовка.
- Проверка Content-Type, Location и других служебных полей.
- Обработка HTTP-ответов по именованным заголовкам.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    v_val VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    v_val := UTL_HTTP_GET_HEADER_BY_NAME(resp_h, 'Content-Type', 1);

    IF v_val IS NULL OR v_val::TEXT = '' THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_HEADER_BY_NAME failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_HEADER_BY_NAME: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.READ_TEXT

Функция читает часть тела HTTP-ответа в текстовом виде.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.
- len (INTEGER) – максимальное число символов для чтения.

Выходные параметры:

- Возвращает фрагмент тела ответа в текстовом виде.

Типичные сценарии использования:

- Чтение текстового тела HTTP-ответа.
- Постепенная обработка больших ответов.
- Получение HTML или JSON как текста.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    res_text VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    res_text := UTL_HTTP_READ_TEXT(resp_h, 100);

    IF res_text IS NULL OR res_text::TEXT = '' THEN
        RAISE EXCEPTION 'UTL_HTTP_READ_TEXT failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_READ_TEXT: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.READ_LINE

Функция читает одну строку из тела HTTP-ответа.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.
- remove_crlf (BOOLEAN) – признак удаления завершающих символов перевода строки.

Выходные параметры:

- Возвращает одну строку из тела ответа.

Типичные сценарии использования:

- Построчное чтение текстовых HTTP-ответов.
- Обработка потоковых текстовых данных.
- Разбор простых текстовых протоколов поверх HTTP.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    line VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    line := UTL_HTTP_READ_LINE(resp_h, TRUE);

    IF line IS NULL OR line::TEXT = '' THEN
        RAISE EXCEPTION 'UTL_HTTP_READ_LINE failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_READ_LINE: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.READ_RAW

Функция читает часть тела HTTP-ответа в бинарном виде.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.
- len (INTEGER) – максимальное число байт для чтения.

Выходные параметры:

- Возвращает бинарные данные ответа.

Типичные сценарии использования:

- Чтение бинарного содержимого HTTP-ответа.
- Получение файлов и двоичных данных по HTTP.
- Обработка содержимого без текстового преобразования.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    res_raw RAW;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    res_raw := UTL_HTTP_READ_RAW(resp_h, 100);

    IF res_raw IS NULL THEN
        RAISE EXCEPTION 'UTL_HTTP_READ_RAW failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_READ_RAW: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.END_RESPONSE

Процедура завершает обработку HTTP-ответа и освобождает связанные ресурсы.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Завершение работы с HTTP-ответом.
- Освобождение сетевых ресурсов.
- Корректное завершение HTTP-сценария.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);

    RAISE NOTICE 'UTL_HTTP_END_RESPONSE: OK';
END;
$$;
```

UTL_HTTP.REQUEST

Функция выполняет простой HTTP-запрос и возвращает тело ответа целиком.

Входящие параметры:

- url (VARCHAR2) – адрес HTTP-ресурса.
- proxy (VARCHAR2) – адрес прокси.
- wallet_path (VARCHAR2) – путь к хранилищу сертификатов.
- wallet_password (VARCHAR2) – пароль хранилища.
- https_host (VARCHAR2) – имя HTTPS-хоста.

Выходные параметры:

- Возвращает тело HTTP-ответа в текстовом виде.

Типичные сценарии использования:

- Быстрое выполнение простого HTTP GET-запроса.
- Получение содержимого ресурса без ручного управления дескрипторами.
- Проверка доступности HTTP-ресурса.

Примеры запросов:

```
DO $$
DECLARE
    res_text VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    PERFORM UTL_HTTP_SET_TRANSFER_TIMEOUT(15);

    -- Тест
    res_text := UTL_HTTP_REQUEST('http://ya.ru'::VARCHAR2, NULL, NULL, NULL, NULL);

    IF res_text IS NULL OR res_text::TEXT = '' THEN
        RAISE EXCEPTION 'UTL_HTTP_REQUEST failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_REQUEST: OK';
END;
$$;
```

UTL_HTTP.WRITE_TEXT

Процедура записывает текстовые данные в тело HTTP-запроса.

Входящие параметры:

- req (INTEGER) – дескриптор HTTP-запроса.
- data (VARCHAR2) – текстовые данные для отправки.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Отправка текстового тела HTTP POST/PUT-запроса.
- Передача JSON, XML и других текстовых форматов.
- Формирование пользовательского HTTP-запроса.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'POST'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    PERFORM UTL_HTTP_SET_HEADER(req_h, 'Content-Type', 'text/plain');

    -- Тест
    PERFORM UTL_HTTP_WRITE_TEXT(req_h, 'Simple text data');

    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    RAISE NOTICE 'UTL_HTTP_WRITE_TEXT: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.WRITE_LINE

Процедура записывает строку с переводом строки в тело HTTP-запроса.

Входящие параметры:

- req (INTEGER) – дескриптор HTTP-запроса.
- data (VARCHAR2) – строка для записи.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Формирование строкового тела запроса.
- Передача данных с разделителями строк.
- Пошаговая запись текстового содержимого.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'POST'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    PERFORM UTL_HTTP_SET_HEADER(req_h, 'Content-Type', 'text/plain');

    -- Тест
    PERFORM UTL_HTTP_WRITE_LINE(req_h, 'Line with CRLF');

    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    RAISE NOTICE 'UTL_HTTP_WRITE_LINE: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.WRITE_RAW

Процедура записывает бинарные данные в тело HTTP-запроса.

Входящие параметры:

- req (INTEGER) – дескриптор HTTP-запроса.
- data (RAW) – бинарные данные для отправки.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Отправка бинарных данных в HTTP-запросе.
- Передача файлов и двоичного содержимого.
- Формирование тела запроса без текстового преобразования.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'POST'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    PERFORM UTL_HTTP_SET_HEADER(req_h, 'Content-Type', 'application/octet-stream');

    -- Тест
    PERFORM UTL_HTTP_WRITE_RAW(req_h, '\x48656c6c6f'::BYTEA);

    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    RAISE NOTICE 'UTL_HTTP_WRITE_RAW: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.SET_AUTHENTICATION

Процедура задаёт параметры HTTP-аутентификации для запроса.

Входящие параметры:

- req (INTEGER) – дескриптор HTTP-запроса.
- username (VARCHAR2) – имя пользователя.
- password (VARCHAR2) – пароль.
- scheme (VARCHAR2) – схема аутентификации.
- for_proxy (BOOLEAN) – признак использования аутентификации для прокси.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Подготовка авторизованного HTTP-запроса.

- Настройка Basic-аутентификации.
- Передача учётных данных серверу или прокси.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);

    -- Тест
    PERFORM UTL_HTTP_SET_AUTHENTICATION(req_h, 'test_user', 'test_password', 'Basic',
FALSE);

    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    RAISE NOTICE 'UTL_HTTP_SET_AUTHENTICATION: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.GET_AUTHENTICATION

Функция возвращает сведения об аутентификации из HTTP-ответа.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.
- for_proxy (BOOLEAN) – признак чтения параметров прокси-аутентификации.

Выходные параметры:

- Возвращает схему и область аутентификации.

Типичные сценарии использования:

- Анализ требований к HTTP-аутентификации.

- Обработка ответов сервера с требованием авторизации.
- Диагностика параметров аутентификации.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    v_scheme VARCHAR2;
    v_realm VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    PERFORM UTL_HTTP_SET_AUTHENTICATION(req_h, 'test_user', 'test_password', 'Basic',
FALSE);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    SELECT scheme, realm INTO v_scheme, v_realm FROM
UTL_HTTP_GET_AUTHENTICATION(resp_h, FALSE);

    RAISE NOTICE 'UTL_HTTP_GET_AUTHENTICATION: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.SET_BODY_CHARSET_RESP

Процедура задаёт кодировку для интерпретации тела HTTP-ответа.

Входящие параметры:

- resp (INTEGER) – дескриптор HTTP-ответа.
- charset (VARCHAR2) – кодировка ответа.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Принудительное задание кодировки ответа.
- Корректная обработка текста при отсутствии нужного заголовка.
- Чтение HTTP-ответов в нестандартной кодировке.

Примеры запросов:

```
DO $$
DECLARE
    req_h INTEGER;
    resp_h INTEGER;
    res_text VARCHAR2;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    req_h := UTL_HTTP_BEGIN_REQUEST('http://ya.ru'::VARCHAR2, 'GET'::VARCHAR2,
'HTTP/1.1'::VARCHAR2, NULL, NULL);
    resp_h := UTL_HTTP_GET_RESPONSE(req_h, FALSE);

    -- Тест
    PERFORM UTL_HTTP_SET_BODY_CHARSET_RESP(resp_h, 'UTF-8');

    res_text := UTL_HTTP_READ_TEXT(resp_h, 10);

    RAISE NOTICE 'UTL_HTTP_SET_BODY_CHARSET_RESP: OK';

    -- Очистка
    PERFORM UTL_HTTP_END_RESPONSE(resp_h);
END;
$$;
```

UTL_HTTP.CLEAR_COOKIES

Процедура очищает cookie, связанные с указанным контекстом или сессией.

Входящие параметры:

- ctx (INTEGER) – идентификатор контекста запросов.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Сброс состояния HTTP-сессии.
- Очистка накопленных cookie перед тестами.
- Изоляция сценариев работы с HTTP.

Примеры запросов:

```
DO $$
BEGIN
    -- Тест
    PERFORM UTL_HTTP_CLEAR_COOKIES(NULL::INTEGER);

    IF UTL_HTTP_GET_COOKIE_COUNT(NULL::INTEGER) != 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_CLEAR_COOKIES failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_CLEAR_COOKIES: OK';
END;
$$;
```

UTL_HTTP.GET_COOKIE_COUNT

Функция возвращает количество cookie в указанном контексте или сессии.

Входящие параметры:

- ctx (INTEGER) – идентификатор контекста запросов.

Выходные параметры:

- Возвращает число сохранённых cookie.

Типичные сценарии использования:

- Проверка состояния cookie-хранилища.
- Контроль результата очистки cookie.
- Мониторинг клиентского HTTP-состояния.

Примеры запросов:

```
DO $$
DECLARE
    v_count INTEGER;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_CLEAR_COOKIES(NULL::INTEGER);

    -- Тест
    v_count := UTL_HTTP_GET_COOKIE_COUNT(NULL::INTEGER);

    IF v_count != 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_COOKIE_COUNT failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_COOKIE_COUNT: OK';
END;
$$;
```

UTL_HTTP.GET_COOKIES

Функция возвращает набор cookie из указанного контекста или сессии.

Входящие параметры:

- ctx (INTEGER) – идентификатор контекста запросов.

Выходные параметры:

- Возвращает набор строк с описанием cookie.

Типичные сценарии использования:

- Перебор всех сохранённых cookie.
- Анализ состояния HTTP-сессии.
- Отладка обработки cookie.

Примеры запросов:

```
DO $$
DECLARE
    v_cookie_rows INTEGER := 0;
    r_cookie RECORD;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_CLEAR_COOKIES(NULL::INTEGER);

    -- Тест
    FOR r_cookie IN SELECT * FROM UTL_HTTP_GET_COOKIES(NULL::INTEGER)
    LOOP
        v_cookie_rows := v_cookie_rows + 1;
    END LOOP;

    IF v_cookie_rows != 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_COOKIES failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_COOKIES: OK';
END;
$$;
```

UTL_HTTP.GET_DETAILED_SQLCODE

Функция возвращает детализированный SQLCODE последней ошибки UTL_HTTP.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает код последней детализированной ошибки.

Типичные сценарии использования:

- Диагностика ошибок HTTP-вызовов.
- Получение дополнительной информации после исключений.
- Анализ причин отказа сетевых операций.

Примеры запросов:

```
DO $$
DECLARE
    v_sqlcode INTEGER;
BEGIN
    -- Тест
    v_sqlcode := UTL_HTTP_GET_DETAILED_SQLCODE();

    IF v_sqlcode != 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_DETAILED_SQLCODE failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_DETAILED_SQLCODE: OK';
END;
$$;
```

UTL_HTTP.GET_DETAILED_SQLERRM

Функция возвращает детализированное сообщение об ошибке последнего вызова UTL_HTTP.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает текст последней детализированной ошибки.

Типичные сценарии использования:

- Диагностика HTTP-исключений.
- Получение текста ошибки после сбоя сетевой операции.
- Логирование причин проблем при HTTP-вызовах.

Примеры запросов:

```
DO $$
DECLARE
    v_sqlerrm VARCHAR2;
BEGIN
    -- Тест
    v_sqlerrm := UTL_HTTP_GET_DETAILED_SQLERRM();

    IF v_sqlerrm::TEXT != '' THEN
        RAISE EXCEPTION 'UTL_HTTP_GET_DETAILED_SQLERRM failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_GET_DETAILED_SQLERRM: OK';
END;
$$;
```

UTL_HTTP.GET_DETAILED_EXCP_SUPPORT

Функция возвращает признак поддержки детализированных исключений UTL_HTTP.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает TRUE или FALSE.

Типичные сценарии использования:

- Проверка доступности режима детализированных ошибок.
- Диагностика возможностей реализации пакета.
- Отладка обработки исключений.

Примеры запросов:

```
DO $$
DECLARE
    v_detailed BOOLEAN;
BEGIN
    -- Тест
    v_detailed := UTL_HTTP_GET_DETAILED_EXCP_SUPPORT();

    RAISE NOTICE 'UTL_HTTP_GET_DETAILED_EXCP_SUPPORT: %', v_detailed;

    -- Очистка
END;
$$;
```

UTL_HTTP.REQUEST_PIECES

Функция выполняет HTTP-запрос и возвращает ответ частями.

Входящие параметры:

- url (VARCHAR2) – адрес HTTP-ресурса.
- max_pieces (INTEGER) – максимальное число возвращаемых частей.
- proxy (VARCHAR2) – адрес прокси.
- wallet_path (VARCHAR2) – путь к хранилищу сертификатов.
- wallet_password (VARCHAR2) – пароль хранилища.
- https_host (VARCHAR2) – имя HTTPS-хоста.

Выходные параметры:

- Возвращает набор частей HTTP-ответа.

Типичные сценарии использования:

- Постепенное чтение больших HTTP-ответов.
- Ограничение объёма получаемых данных.
- Потоковая обработка содержимого ресурса.

Примеры запросов:

```
DO $$
DECLARE
    v_piece_count INTEGER := 0;
    v_piece RECORD;
BEGIN
    -- Подготовка
    PERFORM UTL_HTTP_SET_PROXY(NULL::VARCHAR2, NULL::VARCHAR2);
    PERFORM UTL_HTTP_SET_TRANSFER_TIMEOUT(15);

    -- Тест
    FOR v_piece IN SELECT * FROM UTL_HTTP_REQUEST_PIECES('http://ya.ru'::VARCHAR2, 3,
NULL, NULL, NULL, NULL)
    LOOP
        v_piece_count := v_piece_count + 1;
    END LOOP;

    IF v_piece_count <= 0 THEN
        RAISE EXCEPTION 'UTL_HTTP_REQUEST_PIECES failed';
    END IF;

    RAISE NOTICE 'UTL_HTTP_REQUEST_PIECES: OK';
END;
$$;
```

UTL_HTTP_REQUEST_EXTENDED

Вспомогательная функция для: utl_http_request

Описание: выполняет HTTP-запрос с расширенными параметрами.

Входящие параметры:

- url (VARCHAR2) – URL запроса
- proxy (VARCHAR2) – прокси-сервер (опционально)
- wallet_path (VARCHAR2) – путь к хранилищу сертификатов (опционально)
- wallet_password (VARCHAR2) – пароль хранилища (опционально)
- https_host (VARCHAR2) – HTTPS хост (опционально)
- request_profile (INT4) – профиль запроса

Выходные параметры:

- VARCHAR2 – ответ сервера

Типичные сценарии использования:

- Выполнение HTTP-запросов с настройками
- Работа через прокси и с HTTPS

Примеры запросов:

```
SELECT utl_http_request_extended('http://example.com',NULL,NULL,NULL,NULL,1);
```

UTL_HTTP_GET_VERSION

Функция возвращает номер версии пакета UTL_HTTP для HTTP-запросов.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для веб-запросов
- Совместимость протоколов

Примеры запросов:

```
SELECT utl_http_get_version();
```

5.2.46. Пакет UTL_COMPRESS

UTL_COMPRESS.LZ_COMPRESS

Функция выполняет LZ-сжатие данных.

Входящие параметры:

- src (RAW/BLOB) – исходные данные для сжатия.
- quality (INTEGER) – уровень сжатия.

Выходные параметры:

- Возвращает сжатые данные.

Типичные сценарии использования:

- Сжатие бинарных данных перед хранением или передачей.
- Подготовка компактного представления данных.
- Проверка работы алгоритма LZ-сжатия.

Примеры запросов:

```
DO $$
DECLARE
    v_src RAW := '\x48656c6c6f'::RAW;
    v_cmp RAW;
    v_dec RAW;
BEGIN
    v_cmp := UTL_COMPRESS_LZ_COMPRESS(v_src, 6);
    v_dec := UTL_COMPRESS_LZ_UNCOMPRESS(v_cmp);

    RAISE NOTICE 'Исходные: %', v_src;
    RAISE NOTICE 'Сжатые: % (было % байт, стало % байт)', v_cmp, LENGTH(v_src),
LENGTH(v_cmp);

    IF v_dec::TEXT = v_src::TEXT THEN
        RAISE NOTICE ' Целостность сохранена';
    ELSE
        RAISE EXCEPTION ' Ошибка';
    END IF;

    RAISE NOTICE 'UTL_COMPRESS_LZ_COMPRESS: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.LZ_UNCOMPRESS

Функция выполняет распаковку данных, ранее сжатых алгоритмом LZ.

Входящие параметры:

- src (RAW/BLOB) – сжатые данные.

Выходные параметры:

- Возвращает распакованные данные.

Типичные сценарии использования:

- Восстановление исходных данных после сжатия.
- Проверка целостности данных после распаковки.
- Обработка ранее сжатых бинарных значений.

Примеры запросов:

```
DO $$
DECLARE
    v_src RAW := '\x48656c6c6f'::RAW;
    v_cmp RAW;
    v_dec RAW;
BEGIN
    v_cmp := UTL_COMPRESS_LZ_COMPRESS(v_src, 6);
    v_dec := UTL_COMPRESS_LZ_UNCOMPRESS(v_cmp);

    RAISE NOTICE 'Сжатые: %', v_cmp;
    RAISE NOTICE 'Распакованные: %', v_dec;

    IF v_dec::TEXT = v_src::TEXT THEN
        RAISE NOTICE '□ Распаковка успешна';
    ELSE
        RAISE EXCEPTION '□ Распаковка не совпадает';
    END IF;

    RAISE NOTICE 'UTL_COMPRESS_LZ_UNCOMPRESS: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.LZ_COMPRESS_OPEN

Функция открывает потоковое LZ-сжатие и возвращает хэндл сессии сжатия.

Входящие параметры:

- dst (BLOB) – буфер назначения.
- quality (INTEGER) – уровень сжатия.

Выходные параметры:

- Возвращает хэндл потока сжатия.

Типичные сценарии использования:

- Инициализация потокового сжатия больших объёмов данных.
- Подготовка к пошаговой записи чанков в поток сжатия.
- Организация многошаговой обработки бинарных данных.

Примеры запросов:

```
DO $$
DECLARE
    v_handle INTEGER;
    v_dst BLOB := '\x'::BLOB;
BEGIN
    v_handle := UTL_COMPRESS_LZ_COMPRESS_OPEN(v_dst, 6);
    RAISE NOTICE 'Хэндл создан: %', v_handle;

    IF v_handle IS NOT NULL AND v_handle > 0 THEN
        RAISE NOTICE '□ Открытие потока успешно';
    ELSE
        RAISE EXCEPTION '□ Ошибка открытия';
    END IF;

    PERFORM UTL_COMPRESS_LZ_COMPRESS_CLOSE(v_handle, v_dst);
    RAISE NOTICE 'UTL_COMPRESS_LZ_COMPRESS_OPEN: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.LZ_COMPRESS_ADD

Процедура добавляет очередной фрагмент данных в открытый поток LZ-сжатия.

Входящие параметры:

- handle (INTEGER) – хэндл потока сжатия.
- dst (BLOB) – буфер назначения.
- chunk (RAW) – очередной фрагмент данных.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Пошаговое сжатие данных по частям.
- Обработка больших наборов бинарных данных.
- Формирование итогового сжатого потока из чанков.

Примеры запросов:

```
DO $$
DECLARE
    v_handle INTEGER;
    v_dst BLOB := '\x'::BLOB;
    v_chunk RAW := '\x616263'::RAW;
BEGIN
    v_handle := UTL_COMPRESS_LZ_COMPRESS_OPEN(v_dst, 6);
    PERFORM UTL_COMPRESS_LZ_COMPRESS_ADD(v_handle, v_dst, v_chunk);

    RAISE NOTICE 'Добавлен чанк: %', v_chunk;
    RAISE NOTICE '□ Добавление данных в поток успешно';

    PERFORM UTL_COMPRESS_LZ_COMPRESS_CLOSE(v_handle, v_dst);
    RAISE NOTICE 'UTL_COMPRESS_LZ_COMPRESS_ADD: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.LZ_COMPRESS_CLOSE

Функция завершает потоковое LZ-сжатие и возвращает итоговый результат.

Входящие параметры:

- handle (INTEGER) – хэндл потока сжатия.
- dst (BLOB) – буфер назначения.

Выходные параметры:

- Возвращает итоговый сжатый BLOB.

Типичные сценарии использования:

- Завершение потокового сжатия.
- Получение итогового результата после добавления всех чанков.

- Освобождение ресурсов, связанных с потоком сжатия.

Примеры запросов:

```
DO $$
DECLARE
    v_handle INTEGER;
    v_dst BLOB := '\x'::BLOB;
    v_result BLOB;
BEGIN
    v_handle := UTL_COMPRESS_LZ_COMPRESS_OPEN(v_dst, 6);
    v_result := UTL_COMPRESS_LZ_COMPRESS_CLOSE(v_handle, v_dst);

    RAISE NOTICE 'Результат сжатия: %', v_result;

    IF v_result IS NOT NULL THEN
        RAISE NOTICE '□ Закрытие потока и сжатие успешны';
    ELSE
        RAISE EXCEPTION '□ Ошибка';
    END IF;

    RAISE NOTICE 'UTL_COMPRESS_LZ_COMPRESS_CLOSE: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.LZ_UNCOMPRESS_OPEN

Функция открывает поток распаковки LZ-данных и возвращает его хэндл.

Входящие параметры:

- src (BLOB) – сжатые данные.

Выходные параметры:

- Возвращает хэндл потока распаковки.

Типичные сценарии использования:

- Подготовка к поэтапной распаковке данных.
- Инициализация потока чтения сжатого содержимого.
- Обработка больших сжатых объектов частями.

Примеры запросов:

```
DO $$
DECLARE
    v_handle INTEGER;
    v_src BLOB;
BEGIN
    v_src := UTL_COMPRESS_LZ_COMPRESS('\x44617461'::BLOB, 6);
    v_handle := UTL_COMPRESS_LZ_UNCOMPRESS_OPEN(v_src);

    RAISE NOTICE 'Хэндл распаковки: %', v_handle;

    IF v_handle IS NOT NULL AND v_handle > 0 THEN
        RAISE NOTICE '□ Открытие потока распаковки успешно';
    ELSE
        RAISE EXCEPTION '□ Ошибка';
    END IF;

    PERFORM UTL_COMPRESS_LZ_UNCOMPRESS_CLOSE(v_handle);
    RAISE NOTICE 'UTL_COMPRESS_LZ_UNCOMPRESS_OPEN: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.LZ_UNCOMPRESS_EXTRACT

Функция извлекает очередной фрагмент данных из открытого потока распаковки.

Входящие параметры:

- handle (INTEGER) – хэндл потока распаковки.
- len (INTEGER) – максимальный размер извлекаемого фрагмента.

Выходные параметры:

- Возвращает распакованный фрагмент данных.

Типичные сценарии использования:

- Пошаговая распаковка больших данных.
- Чтение содержимого из потока распаковки по частям.
- Проверка целостности данных после извлечения.

Примеры запросов:

```
DO $$
DECLARE
    v_handle INTEGER;
    v_src BLOB;
    v_chunk RAW;
    v_original RAW := '\x4461746154657374'::RAW;
BEGIN
    v_src := UTL_COMPRESS_LZ_COMPRESS(v_original::BLOB, 6);
    v_handle := UTL_COMPRESS_LZ_UNCOMPRESS_OPEN(v_src);
    v_chunk := UTL_COMPRESS_LZ_UNCOMPRESS_EXTRACT(v_handle, 8192);

    RAISE NOTICE 'Извлеченные данные: %', v_chunk;

    IF v_chunk::TEXT = v_original::TEXT THEN
        RAISE NOTICE '□ Извлечение данных успешно, целостность сохранена';
    ELSE
        RAISE EXCEPTION '□ Данные не совпадают';
    END IF;

    PERFORM UTL_COMPRESS_LZ_UNCOMPRESS_CLOSE(v_handle);
    RAISE NOTICE 'UTL_COMPRESS_LZ_UNCOMPRESS_EXTRACT: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.LZ_UNCOMPRESS_CLOSE

Процедура закрывает поток распаковки и освобождает связанные ресурсы.

Входящие параметры:

- handle (INTEGER) – хэндл потока распаковки.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Корректное завершение потоковой распаковки.
- Освобождение ресурсов после чтения данных.
- Завершение многошагового сценария распаковки.

Примеры запросов:

```
DO $$
DECLARE
    v_handle INTEGER;
    v_src BLOB;
BEGIN
    v_src := UTL_COMPRESS_LZ_COMPRESS('\x44617461'::BLOB, 6);
    v_handle := UTL_COMPRESS_LZ_UNCOMPRESS_OPEN(v_src);
    PERFORM UTL_COMPRESS_LZ_UNCOMPRESS_CLOSE(v_handle);

    RAISE NOTICE '□ Поток распаковки закрыт';
    RAISE NOTICE 'UTL_COMPRESS_LZ_UNCOMPRESS_CLOSE: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS.ISOPEN

Функция проверяет, открыт ли поток сжатия или распаковки по указанному хэндлу.

Входящие параметры:

- handle (INTEGER) – хэндл потока сжатия или распаковки.

Выходные параметры:

- Возвращает `TRUE`, если поток открыт, иначе `FALSE`.

Типичные сценарии использования:

- Контроль состояния потоков сжатия и распаковки.
- Проверка корректности закрытия хэндлов.
- Отладка многошаговых сценариев работы с пакетами сжатия.

Примеры запросов:

```
DO $$
DECLARE
    v_handle INTEGER;
    v_dst BLOB := '\x'::BLOB;
    v_is_open BOOLEAN;
BEGIN
    v_handle := UTL_COMPRESS_LZ_COMPRESS_OPEN(v_dst, 6);
    v_is_open := UTL_COMPRESS_ISOPEN(v_handle);
    RAISE NOTICE 'Хэндл % открыт: %', v_handle, v_is_open;

    PERFORM UTL_COMPRESS_LZ_COMPRESS_CLOSE(v_handle, v_dst);
    v_is_open := UTL_COMPRESS_ISOPEN(v_handle);
    RAISE NOTICE 'После закрытия: %', v_is_open;

    IF v_is_open = FALSE THEN
        RAISE NOTICE '□ Функция ISOPEN корректно определяет состояние';
    ELSE
        RAISE EXCEPTION '□ Ошибка';
    END IF;

    RAISE NOTICE 'UTL_COMPRESS_ISOPEN: УСПЕШНО';
END;
$$;
```

UTL_COMPRESS_LZ_UNCOMPRESS_RAW_COMMON

Вспомогательная функция для: utl_compress_lz_uncompress

Описание: выполняет распаковку LZ-данных с учетом поведения для NULL.

Входящие параметры:

- str (RAW) – сжатые данные
- onempty (BOOLEAN) – поведение при пустых данных

Выходные параметры:

- RAW – распакованные данные

Типичные сценарии использования:

- Распаковка LZ-сжатых данных
- Обработка NULL и пустых значений

Примеры запросов:

```
SELECT utl_compress_lz_uncompress_raw_common(NULL, false);
```

UTL_COMPRESS_GET_VERSION

Функция возвращает номер версии пакета UTL_COMPRESS для сжатия данных.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии алгоритмов сжатия
- Совместимость архивов

Примеры запросов:

```
SELECT utl_compress_get_version();
```

5.2.47. Пакет UTL_FILE

Ниже приведены описания функций и процедур пакета `UTL_FILE` с примерами запросов. Перед выполнением примеров необходимо подготовить окружение.

Подготовка окружения:

```
-- Подготовка окружения
CREATE TABLE IF NOT EXISTS utl_file_dir (
    dirname text PRIMARY KEY,
    dir text UNIQUE
);
DELETE FROM utl_file_dir
WHERE dirname = 'UTL_TEST_DIR';
INSERT INTO utl_file_dir (dirname, dir)
VALUES ('UTL_TEST_DIR', '/tmp')
ON CONFLICT (dirname) DO UPDATE
SET dir = EXCLUDED.dir;
SELECT 'UTL_FILE ENVIRONMENT READY' AS status;
```

UTL_FILE.FOPEN

Функция открывает файл в заданной директории и возвращает файловый дескриптор.

Входящие параметры:

- location (VARCHAR2) – имя логической директории.
- filename (VARCHAR2) – имя файла.
- open_mode (VARCHAR2) – режим открытия файла.

Выходные параметры:

- Возвращает файловый дескриптор.

Типичные сценарии использования:

- Открытие файла для чтения или записи.
- Подготовка к дальнейшим операциям ввода-вывода.
- Проверка доступности каталога и файла.
- В тестовом сценарии ожидается результат: true, если файл успешно открыт.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fopen.txt', 'w');
    RAISE INFO 'Result utl_file_fopen: %', utl_file_is_open(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.IS_OPEN

Функция проверяет, открыт ли файл по указанному дескриптору.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.

Выходные параметры:

- Возвращает `TRUE`, если файл открыт, иначе `FALSE`.

Типичные сценарии использования:

- Контроль состояния файлового дескриптора.
- Проверка корректности открытия или закрытия файла.
- Использование в тестах операций `UTL_FILE`.
- В тестовом сценарии ожидается результат: true для открытого файла.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_is_open.txt', 'w');
    RAISE INFO 'Result utl_file_is_open: %', utl_file_is_open(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.PUT_LINE

Процедура записывает строку в файл и добавляет перевод строки.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.
- buffer (VARCHAR2) – записываемая строка.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Запись текстовых строк в файл.
- Формирование построчного файла.
- Логирование и экспорт данных в текстовом виде.
- В тестовом сценарии ожидается результат: строка, записанная в файл.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_put_line.txt', 'w');
    PERFORM utl_file_put_line(h, 'DATA_LINE');
    PERFORM utl_file_fclose(h);
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_put_line.txt', 'r');
    RAISE INFO 'Result utl_file_put_line: %', utl_file_get_line(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.GET_LINE

Функция считывает одну строку из файла.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.

Выходные параметры:

- Возвращает считанную строку.

Типичные сценарии использования:

- Чтение текстовых данных из файла.
- Построчная обработка содержимого файла.
- Проверка ранее записанных значений.
- В тестовом сценарии ожидается результат: считанная строка.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_get_line.txt', 'w');
    PERFORM utl_file_put_line(h, 'GET_LINE_DATA');
    PERFORM utl_file_fclose(h);
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_get_line.txt', 'r');
    RAISE INFO 'Result utl_file_get_line: %', utl_file_get_line(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.FFLUSH

Процедура принудительно сбрасывает буфер записи файла на диск.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Гарантированная запись буферизованных данных.
- Промежуточная фиксация данных в файле.
- Контроль длины файла после записи.
- В тестовом сценарии ожидается результат: длина файла после сброса буфера.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
    v_len bigint;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fflush.txt', 'w');
    PERFORM utl_file_put(h, 'FLUSH_DATA');
    PERFORM utl_file_fflush(h);
    SELECT file_length
      INTO v_len
      FROM utl_file_fgetattr('UTL_TEST_DIR', 'doc_fflush.txt');
    RAISE INFO 'Result utl_file_fflush: %', v_len;
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.FCLOSE

Процедура закрывает открытый файл.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Завершение работы с файлом.
- Освобождение файловых ресурсов.
- Проверка изменения состояния дескриптора после закрытия.
- В тестовом сценарии ожидается результат: false после закрытия файла.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fclose.txt', 'w');
    PERFORM utl_file_fclose(h);
    RAISE INFO 'Result utl_file_fclose: %', utl_file_is_open(h);
END $$;
```

UTL_FILE.PUT

Процедура записывает текст в файл без добавления перевода строки.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.
- buffer (VARCHAR2) – записываемый текст.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Пошаговое формирование строки в файле.
- Запись данных без автоматического завершения строки.
- Формирование текстового содержимого по частям.
- В тестовом сценарии ожидается результат: объединенный текст без перевода строки.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_put.txt', 'w');
    PERFORM utl_file_put(h, 'A');
    PERFORM utl_file_put(h, 'B');
    PERFORM utl_file_fclose(h);
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_put.txt', 'r');
    RAISE INFO 'Result utl_file_put: %', utl_file_get_line(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.PUTF

Процедура записывает в файл форматированную строку.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.
- format (VARCHAR2) – строка формата.
- args – значения для подстановки в формат.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Форматированная запись значений в файл.
- Построение шаблонных строк.
- Экспорт параметризованных текстовых данных.
- В тестовом сценарии ожидается результат: форматированная строка.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_putf.txt', 'w');
    PERFORM utl_file_putf(h, 'V:%s', '1');
    PERFORM utl_file_fclose(h);
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_putf.txt', 'r');
    RAISE INFO 'Result utl_file_putf: %', utl_file_get_line(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.NEW_LINE

Процедура добавляет один или несколько переводов строки в файл.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.
- lines (INTEGER) – число добавляемых переводов строки.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Разделение строк в файле.
- Формирование многострочного текстового содержимого.
- Управление форматированием текстового вывода.
- В тестовом сценарии ожидается результат: две отдельные строки в файле.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
    v_line1 text;
    v_line2 text;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_new_line.txt', 'w');
    PERFORM utl_file_put(h, 'A');
    PERFORM utl_file_new_line(h, 1);
    PERFORM utl_file_put(h, 'B');
    PERFORM utl_file_fclose(h);
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_new_line.txt', 'r');
    v_line1 := utl_file_get_line(h);
    v_line2 := utl_file_get_line(h);
    PERFORM utl_file_fclose(h);
    RAISE INFO 'Result utl_file_new_line: [%] [%]', v_line1, v_line2;
END $$;
```

UTL_FILE.FGETATTR

Функция возвращает атрибуты файла, включая факт существования и длину.

Входящие параметры:

- location (VARCHAR2) – имя логической директории.
- filename (VARCHAR2) – имя файла.

Выходные параметры:

- Возвращает сведения о файле, включая `fexists` и `file_length`.

Типичные сценарии использования:

- Проверка существования файла.
- Получение длины файла после записи.
- Контроль метаданных файловой системы.
- В тестовом сценарии ожидается результат: файл существует и имеет ожидаемую длину.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fgetattr.txt', 'w');
    PERFORM utl_file_put(h, '12345');
    PERFORM utl_file_fclose(h);
END $$;
```

SELECT fexists, file_length FROM utl_file_fgetattr('UTL_TEST_DIR', 'doc_fgetattr.txt');

UTL_FILE.FCOPY

Процедура копирует файл из одной директории в другую.

Входящие параметры:

- src_location (VARCHAR2) – исходная логическая директория.
- src_filename (VARCHAR2) – имя исходного файла.
- dest_location (VARCHAR2) – целевая логическая директория.
- dest_filename (VARCHAR2) – имя целевого файла.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Копирование подготовленных файлов.
- Создание резервных копий.
- Перемещение данных между логическими директориями через копирование.
- В тестовом сценарии ожидается результат: содержимое скопированного файла.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fcopy_src.txt', 'w');
    PERFORM utl_file_put_line(h, 'COPY_DATA');
    PERFORM utl_file_fclose(h);
    PERFORM utl_file_fcopy(
        'UTL_TEST_DIR',
        'doc_fcopy_src.txt',
        'UTL_TEST_DIR',
        'doc_fcopy_dst.txt'
    );
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fcopy_dst.txt', 'r');
    RAISE INFO 'Result utl_file_fcopy: %', utl_file_get_line(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.FRENAME

Процедура переименовывает файл или перемещает его в другую логическую директорию.

Входящие параметры:

- src_location (VARCHAR2) – исходная логическая директория.
- src_filename (VARCHAR2) – текущее имя файла.
- dest_location (VARCHAR2) – целевая логическая директория.
- dest_filename (VARCHAR2) – новое имя файла.
- overwrite (BOOLEAN) – разрешение на перезапись.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Переименование файлов.
- Перемещение файла внутри файловой подсистемы.
- Организация итоговых имён выходных файлов.
- В тестовом сценарии ожидается результат: содержимое файла после

переименования.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_frename_old.txt', 'w');
    PERFORM utl_file_put_line(h, 'RENAME_DATA');
    PERFORM utl_file_fclose(h);
    PERFORM utl_file_frename(
        'UTL_TEST_DIR',
        'doc_frename_old.txt',
        'UTL_TEST_DIR',
        'doc_frename_new.txt',
        true
    );
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_frename_new.txt', 'r');
    RAISE INFO 'Result utl_file_frename: %', utl_file_get_line(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE.FREMOVE

Процедура удаляет файл из указанной директории.

Входящие параметры:

- location (VARCHAR2) – имя логической директории.
- filename (VARCHAR2) – имя удаляемого файла.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Удаление временных или служебных файлов.
- Очистка результатов тестов.
- Управление жизненным циклом файлов.
- В тестовом сценарии ожидается результат: false, если файл удален.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fremove.txt', 'w');
    PERFORM utl_file_put(h, 'REMOVE_ME');
    PERFORM utl_file_fclose(h);
    PERFORM utl_file_fremove('UTL_TEST_DIR', 'doc_fremove.txt');
    RAISE INFO 'Result utl_file_fremove: %',
        (utl_file_fgetattr('UTL_TEST_DIR', 'doc_fremove.txt')).fexists;
END $$;
```

UTL_FILE.FCLOSE_ALL

Процедура закрывает все открытые файловые дескрипторы текущей сессии.

Входящие параметры:

- Нет.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Аварийное или пакетное закрытие файлов.
- Очистка состояния после серии тестов.
- Освобождение всех файловых ресурсов сессии.
- В тестовом сценарии ожидается результат: false после закрытия всех открытых файлов.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_fclose_all.txt', 'w');
    PERFORM utl_file_fclose_all();
    RAISE INFO 'Result utl_file_fclose_all: %', utl_file_is_open(h);
END $$;
```

UTL_FILE.TMPDIR

Функция возвращает путь к временной директории, используемой пакетом `UTL_FILE`.

Входящие параметры:

- Нет.

Выходные параметры:

- Возвращает путь к временной директории.

Типичные сценарии использования:

- Проверка конфигурации временного каталога.
- Определение расположения временных файлов.
- Диагностика окружения `UTL_FILE`.
- В тестовом сценарии ожидается результат: путь к временной директории.

Примеры запросов:

```
SELECT utl_file_tmpdir() AS result_tmpdir;
```

UTL_FILE.GET_NEXTLINE

Функция считывает следующую строку из файла.

Входящие параметры:

- file (INTEGER) – файловый дескриптор.

Выходные параметры:

- Возвращает следующую строку файла.

Типичные сценарии использования:

- Последовательное чтение строк файла.
- Построчная обработка текстовых данных.
- Проверка содержимого, записанного ранее.
- В тестовом сценарии ожидается результат: следующая строка из файла.

Примеры запросов:

```
DO $$
DECLARE
    h integer;
BEGIN
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_get_nextline.txt', 'w');
    PERFORM utl_file_put_line(h, 'NEXT_DATA');
    PERFORM utl_file_fclose(h);
    h := utl_file_fopen('UTL_TEST_DIR', 'doc_get_nextline.txt', 'r');
    RAISE INFO 'Result utl_file_get_nextline: %', utl_file_get_nextline(h);
    PERFORM utl_file_fclose(h);
END $$;
```

UTL_FILE_GET_LINE_FAST

Вспомогательная функция для: utl_file_get_line

Описание: читает строку из файла с учетом ограничений и режима EOF.

Входящие параметры:

- file (INTEGER) – дескриптор файла
- max_linesize (INTEGER) – максимальная длина строки
- eof_mode (INT4) – режим обработки конца файла

Выходные параметры:

- TEXT – прочитанная строка

Типичные сценарии использования:

- Эффективное чтение строк из файла

- Обработка больших файлов

Примеры запросов:

```
SELECT utl_file_get_line_fast(10,1024,2);
```

UTL_FILE_GET_VERSION

Функция возвращает номер версии пакета UTL_FILE для работы с файлами.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для файловых операций
- Совместимость функций

Примеры запросов:

```
SELECT utl_file_get_version();
```

5.2.48. Пакет UTL_I18N

UTL_I18N.STRING_TO_RAW

Функция преобразует строку (VARCHAR2) в тип данных RAW с использованием указанной кодировки символов.

Входящие параметры:

str (VARCHAR2) – исходная строка.

charset (VARCHAR2) – целевая кодировка символов (например, 'AL32UTF8', 'CL8MSWIN1251').

Выходные параметры:

- Возвращает двоичное представление строки в заданной кодировке.

Типичные сценарии использования:

- Подготовка строковых данных для криптографических функций (например, DBMS_CRYPTO), которые работают с RAW.
- Конвертация текста перед отправкой во внешние системы с определенной кодировкой.
- Фиксация кодировки строки для последующей однозначной интерпретации.

Примеры запросов:

```
select utl_i18n_string_to_raw('é', 'WE8ISO8859P1');
```

UTL_I18N.RAW_TO_CHAR

Функция преобразует двоичные данные (RAW) в строку (VARCHAR2), используя указанную кодировку символов.

Входящие параметры:

- r (RAW) – двоичные данные.
- charset (VARCHAR2) – кодировка символов, в которой интерпретировать RAW-данные.

Выходные параметры:

- Возвращает декодированная строка.

Типичные сценарии использования:

- Преобразование результатов криптографических операций (шифрование, хеширование) обратно в читаемый текст.
- Чтение текстовых данных, полученных из внешних источников в виде двоичного потока с известной кодировкой.
- Корректное отображение многоязычного контента.

Примеры запросов:

```
select utl_i18n_raw_to_char('\x616263646566C2AA'::RAW, 'utf8');
```

UTL_I18N.VALIDATE_CHARACTER_ENCODING

Процедура проверяет, может ли строка быть корректно преобразована в указанную кодировку символов.

Входящие параметры:

- str (VARCHAR2) – строка для проверки.
- charset (VARCHAR2) – кодировка символов для валидации.

Выходные параметры:

- Процедура не возвращает значения. В случае невозможности преобразования возникает исключение.

Типичные сценарии использования:

- Предварительная валидация данных перед попыткой их конвертации или сохранения в системе с другой кодировкой.
- Обеспечение целостности данных при миграции или экспорте/импорте.
- Проверка пользовательского ввода на соответствие требуемой кодировке.

Примеры запросов:

```
select utl_i18n_validate_character_encoding('hello é');
```

UTL_I18N.ESCAPE_REFERENCE

Функция преобразует строку в escape-последовательность в формате Unicode (например, \u0410 для кириллической 'А').

Входящие параметры:

- str (VARCHAR2) – исходная строка.
- charset (VARCHAR2) – кодировка, в которую нужно преобразовать escape-последовательности (опционально).

Выходные параметры:

- Возвращает строку с escape-последовательностями.

Типичные сценарии использования:

- Подготовка строк для включения в XML, JSON или другие форматы данных, где требуется escape-кодирование специальных или не-ASCII символов.
- Безопасная передача строк, содержащих символы, которые могут некорректно интерпретироваться протоколом или приложением.
- Хранение сложного текста в упрощенном текстовом окружении.

Примеры запросов:

```
select utl_i18n_escape_reference('A < B', 'US7ASCII');
```

UTL_I18N.UNESCAPE_REFERENCE

Функция преобразует escape-последовательности в строке (например, \u0410) обратно в соответствующие символы.

Входящие параметры:

- str (VARCHAR2) – строка, содержащая escape-последовательности.

Выходные параметры:

- Возвращает строку с восстановленными символами.

Типичные сценарии использования:

- Обработка данных, полученных из внешних систем (веб-сервисы, API), которые используют escape-кодирование.
- Чтение и интерпретация строк из файлов конфигурации или шаблонов, содержащих escape-последовательности.
- Декодирование текста после операций, использовавших ESCAPE_REFERENCE.

Примеры запросов:

```
select utl_i18n_unescape_reference('hello &#x41;');
```

UTL_I18N.GET_DEFAULT_CHARSET

Функция возвращает имя кодировки символов по умолчанию для текущей сессии или базы данных.

Входящие параметры:

- type (PLS_INTEGER) – тип запрашиваемой кодировки по умолчанию (например, для сессии, для базы данных).

Выходные параметры:

- Возвращает имя кодировки по умолчанию (например, 'AL32UTF8').

Типичные сценарии использования:

- Определение текущих настроек кодировки для корректной обработки текстовых данных.
- Логирование или аудит параметров среды выполнения.
- Принятие решений о необходимости конвертации данных.

Примеры запросов:

```
select utl_i18n_get_default_charset('French', 1, false);
select utl_i18n_get_default_charset('French', 1, true);
select utl_i18n_get_default_charset('French', 1, false);
```

UTL_I18N.GET_DEFAULT_ISO_CURRENCY

Функция возвращает код валюты по умолчанию (ISO 4217) для заданной территории (территории базы данных или сессии).

Входящие параметры:

- territory (VARCHAR2) – код территории (например, 'RUSSIA', 'USA'). Если NULL, используется территория по умолчанию.

Выходные параметры:

- Возвращает трехбуквенный код валюты (например, 'RUB', 'USD').

Типичные сценарии использования:

- Определение валюты по умолчанию для форматирования денежных сумм в

отчетах.

- Инициализация параметров финансовых приложений в зависимости от географического контекста.
- Проверка соответствия введенных данных о валюте ожидаемым для территории.

Примеры запросов:

```
select utl_i18n_get_default_iso_currency('CHINA');
```

UTL_I18N.GET_DEFAULT_LINGUISTIC_SORT

Функция возвращает имя лингвистического правила сортировки по умолчанию для указанного языкового контекста.

Входящие параметры:

- locale (VARCHAR2) – строка локали (например, 'RUSSIAN', 'SPANISH').

Выходные параметры:

- Возвращает имя параметра сортировки (например, 'RUSSIAN', 'GENERIC_M').

Типичные сценарии использования:

- Настройка правил сортировки (ORDER BY) для многоязычных данных в запросах.
- Программное определение параметров сортировки для динамически формируемых отчетов.
- Обеспечение корректной алфавитной группировки данных на разных языках.

Примеры запросов:

```
select utl_i18n_get_default_linguistic_sort('Japanese');
```

UTL_I18N.GET_MAX_CHARACTER_SIZE

Функция возвращает максимальный размер в байтах, необходимый для хранения одного символа в указанной кодировке.

Входящие параметры:

- charset (VARCHAR2) – имя кодировки символов.

Выходные параметры:

- Возвращает максимальный размер символа в байтах (например, 1 для WE8MSWIN1252, 4 для AL32UTF8).

Типичные сценарии использования:

- Расчет размера хранилища для текстовых полей с учетом многоязычного контента.
- Оптимизация производительности при обработке больших текстов, когда важны операции с байтами.
- Проверка ограничений внешних систем на длину строк в байтах.

Примеры запросов:

```
select utl_i18n_get_max_character_size('AL32UTF8');
```

UTL_I18N.GET_TRANSLATION

Функция возвращает перевод заданного текстового литерала на указанный язык, используя лингвистические настройки.

Входящие параметры:

- src_str (VARCHAR2) – исходная строка для перевода.
- src_lang (VARCHAR2) – исходный язык строки.
- dst_lang (VARCHAR2) – целевой язык для перевода.

Выходные параметры:

- Возвращает переведенная строка.

Типичные сценарии использования:

- Динамическая локализация сообщений и интерфейсов приложения в зависимости от языка пользователя.
- Автоматический перевод стандартных терминов или названий в отчетах.

- Поддержка многоязычных справочников внутри базы данных.

Примеры запросов:

```
select utl_i18n_get_translation('GERMAN_GERMANY', 'AMERICAN', 2);
```

UTL_I18N.MAP_CHARSET

Функция возвращает каноническое (официальное) имя кодировки символов по ее псевдониму или имени из другой системы.

Входящие параметры:

- charset (VARCHAR2) – имя или псевдоним кодировки.

Выходные параметры:

- Возвращает каноническое имя кодировки.

Типичные сценарии использования:

- Нормализация имен кодировок, полученных из внешних источников (например, из клиентских приложений, файловых заголовков).
- Обеспечение совместимости при миграции данных между разными платформами или версиями СУБД.
- Упрощение логики обработки кодировок путем приведения их к стандартному виду.

Примеры запросов:

```
select utl_i18n_map_charset('WE8DEC', 1, 1);
```

UTL_I18N.MAP_FROM_SHORT_LANGUAGE

Функция преобразует короткий (двухбуквенный) код языка (например, 'ru', 'en') в полное имя языка.

Входящие параметры:

- short_lang (VARCHAR2) – короткий код языка (ISO 639-1).

Выходные параметры:

- Возвращает полное имя языка (например, 'RUSSIAN', 'AMERICAN').

Типичные сценарии использования:

- Конвертация кодов языка, поступающих из веб-приложений или международных стандартов, во внутренний формат для установки сессии (NLS_LANGUAGE).
- Упрощение интеграции с внешними системами, использующими короткие коды.

Примеры запросов:

```
select utl_i18n_map_from_short_language('US');
```

UTL_I18N.MAP_LANGUAGE_FROM_ISO

Функция преобразует код языка стандарта ISO (ISO 639-2/T или B) в имя языка.

Входящие параметры:

- iso_language (VARCHAR2) – код языка по стандарту ISO 639-2 (трехбуквенный).

Выходные параметры:

- Возвращает имя языка.

Типичные сценарии использования:

- Обработка метаданных из международных источников (библиографические системы, архивы), где используются ISO-коды.
- Сопоставление языковых настроек при импорте данных из систем, строго следующих стандартам ISO.

Примеры запросов:

```
select utl_i18n_map_language_from_iso('en_US');
```

UTL_I18N.MAP_TO_SHORT_LANGUAGE

Функция преобразует полное имя языка в короткий (двухбуквенный) код языка (ISO 639-1).

Входящие параметры:

- ora_language (VARCHAR2) – имя языка.

Выходные параметры:

- Возвращает короткий код языка (например, 'ru', 'en').

Типичные сценарии использования:

- Подготовка данных для экспорта во внешние системы, API или веб-сервисы, ожидающие короткие коды языка.
- Генерация URL, заголовков HTTP или других метаданных, требующих стандартных языковых тэгов.

Примеры запросов:

```
select utl_i18n_map_to_short_language('american');
```

UTL_I18N.TRANSLITERATE

Функция преобразует текст из одного письменного представления в другое (транслитерация), например, из кириллицы в латиницу или наоборот.

Входящие параметры:

- str (VARCHAR2) – исходная строка.
- src_script (VARCHAR2) – исходная система письма (например, 'Cyrillic').
- dst_script (VARCHAR2) – целевая система письма (например, 'Latin').

Выходные параметры:

- Возвращает транслитерированную строку.

Типичные сценарии использования:

- Нормализация имен собственных (имен людей, названий мест) для поиска или сопоставления в международных системах.

- Подготовка данных для систем, не поддерживающих определенные алфавиты (например, преобразование кириллических URL в ASCII).
- Создание читаемых вариантов текста для пользователей, не знакомых с оригинальным алфавитом.

Примеры запросов:

```
select utl_i18n_transliterate('[ça va? naïve façade; coöperate; Straße; cœur; œuvre;
æther; élève]', 'latin_ascii_din91379');
```

UTL_I18N.MAP_LOCALE_TO_ISO

Функция преобразует локаль (комбинацию территории и языка) в соответствующую строку локали по стандарту ISO.

Входящие параметры:

- ora_locale (VARCHAR2) – локаль (например, 'RUSSIAN_RUSSIA').

Выходные параметры:

- Возвращает строку локали в формате ISO (например, 'ru_RU').

Типичные сценарии использования:

- Настройка интернационализации (i18n) в приложениях, которые используют стандартные ISO-локали (например, Java, .NET).
- Экспорт данных с метаинформацией о языке и территории в формате, понятном внешним системам.
- Создание конфигурационных файлов или параметров командной строки для утилит, работающих с ISO-локалями.

Примеры запросов:

```
select utl_i18n_map_locale_to_iso('American','America');
```

UTL_I18N_RAW_TO_CHAR_OPT

Вспомогательная функция для: utl_i18n_raw_to_nchar

Описание: преобразует RAW в строку с учетом политики кодировки.

Входящие параметры:

- r (RAW) – исходные RAW-данные
- charset (VARCHAR2) – кодировка
- charset_policy (INT4) – политика обработки кодировки

Выходные параметры:

- VARCHAR2 – преобразованная строка

Типичные сценарии использования:

- Конвертация бинарных данных в текст
- Работа с разными кодировками

Примеры запросов:

```
SELECT utl_i18n_raw_to_char_opt(decode('74657374','hex'),'UTF8',1);
```

UTL_I18N_GET_VERSION

Функция возвращает номер версии пакета UTL_I18N для интернационализации.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для локализации
- Совместимость кодировок

Примеры запросов:

```
SELECT utl_i18n_get_version();
```

5.2.49. Пакет UTL_MAIL

UTL_MAIL.SEND

Процедура отправляет электронное письмо через встроенный почтовый сервер.

Входящие параметры:

- sender (VARCHAR2) – адрес электронной почты отправителя.
- recipients (VARCHAR2) – адрес или список адресов получателей (разделенных запятыми).
- cc (VARCHAR2) – адрес или список адресов для копии письма.
- bcc (VARCHAR2) – адрес или список адресов для скрытой копии.
- subject (VARCHAR2) – тема письма.
- message (VARCHAR2) – текст сообщения.
- mime_type (VARCHAR2) – MIME-тип содержимого.
- priority (PLS_INTEGER) – приоритет письма (1-высокий, 3-нормальный, 5-низкий).
- replyto (VARCHAR2) – адрес для ответа.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Автоматическая отправка уведомлений из триггеров и хранимых процедур.
- Рассылка отчетов и результатов обработки данных.
- Оповещение администраторов о критических событиях в БД.
- Отправка сгенерированных документов и форм.

Примеры запросов:

```
DO $$
BEGIN
    PERFORM utl_mail_send(
        'alerts@company.com',           -- sender
        'user@company.com',             -- recipients
        NULL,                           -- cc
        NULL,                           -- bcc
        'Система работает',             -- subject
        'Проверка UTL_MAIL',            -- message
        'text/plain; charset=UTF-8',    -- mime_type
        NULL,                           -- priority
        NULL                            -- replyto
    );
END $$;
```

UTL_MAIL.SEND_ATTACH_VARCHAR2

Процедура отправляет электронное письмо с текстовым вложением через встроенный почтовый сервер. Вложение передается в виде строки VARCHAR2.

Входящие параметры:

- sender (VARCHAR2) – адрес электронной почты отправителя.
- recipients (VARCHAR2) – адрес или список адресов получателей (разделенных запятыми).
- cc (VARCHAR2) – адрес или список адресов для копии письма.
- bcc (VARCHAR2) – адрес или список адресов для скрытой копии.
- subject (VARCHAR2) – тема письма.
- message (VARCHAR2) – текст сообщения.
- mime_type (VARCHAR2) – MIME-тип содержимого письма.
- priority (PLS_INTEGER) – приоритет письма (1-высокий, 3-нормальный, 5-низкий).
- attachment (VARCHAR2) – содержимое вложения в виде строки VARCHAR2.
- inline (VARCHAR2) – флаг, указывающий является ли вложение встроенным (TRUE) или отдельным файлом (FALSE).
- att_mime_type (VARCHAR2) – MIME-тип вложения.
- att_filename (VARCHAR2) – имя файла вложения.

- att_last_line (VARCHAR2) – дополнительные параметры вложения.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Отправка сгенерированных отчетов в виде текстовых файлов.
- Передача логов и результатов выполнения процедур.
- Рассылка конфигурационных файлов или скриптов.
- Отправка данных в структурированном текстовом формате (CSV, XML, JSON).

Примеры запросов:

```
DO $$
BEGIN
    PERFORM utl_mail_send_attach_varchar2(
        'alerts@company.com',           -- sender
        'user@company.com',             -- recipients
        NULL,                           -- cc
        NULL,                           -- bcc
        'Система работает',             -- subject
        'Проверка UTL_MAIL',            -- message
        'text/plain; charset=UTF-8',    -- mime_type
        NULL,                           -- priority
        'Привет, мир!',                 -- attachment
        TRUE,                           -- att_inline
        'text/plain; charset=UTF-8',    -- att_mime_type
        NULL,                           -- att_filename
        NULL                            -- replyto
    );
END $$;
```

UTL_MAIL.SEND_ATTACH_RAW

Процедура отправляет электронное письмо с двоичным вложением через встроенный почтовый сервер. Вложение передается в формате RAW.

Входящие параметры:

- sender (VARCHAR2) – адрес электронной почты отправителя.

- recipients (VARCHAR2) – адрес или список адресов получателей (разделенных запятыми).
- cc (VARCHAR2) – адрес или список адресов для копии письма.
- bcc (VARCHAR2) – адрес или список адресов для скрытой копии.
- subject (VARCHAR2) – тема письма.
- message (VARCHAR2) – текст сообщения.
- mime_type (VARCHAR2) – MIME-тип содержимого письма.
- priority (PLS_INTEGER) – приоритет письма (1-высокий, 3-нормальный, 5-низкий).
- attachment (RAW) – двоичное содержимое вложения в формате RAW.
- inline (BOOLEAN) – флаг, указывающий является ли вложение встроенным (TRUE) или отдельным файлом (FALSE).
- att_mime_type (VARCHAR2) – MIME-тип вложения.
- att_filename (VARCHAR2) – имя файла вложения.
- att_last_line (VARCHAR2) – дополнительные параметры вложения.

Выходные параметры:

- Процедура не возвращает значения.

Типичные сценарии использования:

- Отправка бинарных файлов: изображений, PDF-документов, архивов.
- Передача сгенерированных отчетов в двоичном формате.
- Рассылка файлов с зашифрованными данными.
- Отправка вложений, содержащих произвольные двоичные данные.

Примеры запросов:

```
DO $$
BEGIN
    PERFORM utl_mail_send_attach_raw(
        'alerts@company.com',           -- sender
        'user@company.com',             -- recipients
        NULL,                           -- cc
        NULL,                           -- bcc
        'Система работает',             -- subject
        'Проверка UTL_MAIL',            -- message
        'text/plain; charset=UTF-8',    -- mime_type
        NULL,                           -- priority
        '\xffffffff'::RAW,              -- attachment
        TRUE,                           -- att_inline
        'text/plain; charset=UTF-8',    -- att_mime_type
        NULL,                           -- att_filename
        NULL                            -- replyto
    );
END $$;
```

UTL_MAIL_GEN_BOUNDARY

Вспомогательная функция для: utl_mail_send, smtp_send

Описание: генерирует boundary для multipart-сообщений.

Входящие параметры:

- Нет параметров

Выходные параметры:

- TEXT – уникальная граница (boundary)

Типичные сценарии использования:

- Формирование MIME-сообщений
- Отправка multipart-писем

Примеры запросов:

```
SELECT utl_mail_gen_boundary();
```

UTL_MAIL_GET_VERSION

Функция возвращает номер версии пакета UTL_MAIL для отправки электронной почты.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для почтовых уведомлений
- Совместимость форматов

Примеры запросов:

```
SELECT utl_mail_get_version();
```

5.2.50. Пакет UTL_REF

Для дальнейшего тестирования работоспособности отдельных функций пакета `UTL_REF` необходимо выполнить следующие предварительные шаги.

Приведённый ниже код подготавливает тестовое окружение: создаёт временную таблицу `utl_ref_demo`, которая будет автоматически удалена в конце сессии, а также вспомогательную функцию `pg_temp.utl_make_ref(p_id integer)`, формирующую значение `REF` для строк этой таблицы в формате текущей реализации.

```
-- Примеры UTL_REF.
-- Временные объекты автоматически удаляются в конце сессии.

DO $$
BEGIN
    PERFORM set_config('client_min_messages', 'warning', true);
    EXECUTE 'DROP TABLE IF EXISTS utl_ref_demo';
    EXECUTE '
        CREATE TEMP TABLE utl_ref_demo (
            id      integer PRIMARY KEY,
            payload text
```

```
        )
    ';
END $$;

-- Вспомогательная функция формирует REF для строки таблицы utl_ref_demo.
-- Формат текущей реализации: relid, тип ключа, длина ключа в байтах, значение ключа.

CREATE OR REPLACE FUNCTION pg_temp.utl_make_ref(p_id integer)
RETURNS raw
LANGUAGE plpgsql
AS $$
DECLARE
    v_relid   integer := 'utl_ref_demo'::regclass::oid;
    v_keytype integer := 'integer'::regtype::oid;
    v_ref     bytea   := decode(repeat('00', 16), 'hex');
BEGIN
    v_ref := set_byte(v_ref, 0,  v_relid        & 255);
    v_ref := set_byte(v_ref, 1, (v_relid >> 8)  & 255);
    v_ref := set_byte(v_ref, 2, (v_relid >> 16) & 255);
    v_ref := set_byte(v_ref, 3, (v_relid >> 24) & 255);

    v_ref := set_byte(v_ref, 4,  v_keytype        & 255);
    v_ref := set_byte(v_ref, 5, (v_keytype >> 8)  & 255);
    v_ref := set_byte(v_ref, 6, (v_keytype >> 16) & 255);
    v_ref := set_byte(v_ref, 7, (v_keytype >> 24) & 255);

    v_ref := set_byte(v_ref, 8,  4);
    v_ref := set_byte(v_ref, 9,  0);
    v_ref := set_byte(v_ref, 10, 0);
    v_ref := set_byte(v_ref, 11, 0);

    v_ref := set_byte(v_ref, 12,  p_id        & 255);
    v_ref := set_byte(v_ref, 13, (p_id >> 8)  & 255);
    v_ref := set_byte(v_ref, 14, (p_id >> 16) & 255);
    v_ref := set_byte(v_ref, 15, (p_id >> 24) & 255);

    RETURN v_ref::raw;
END $$;
```

После выполнения указанных подготовительных действий можно переходить к тестированию отдельных функций и процедур пакета `UTL_REF`.

UTL_REF.DELETE_OBJECT

Процедура удаляет строку по значению `REF`.

Входящие параметры:

- ref (RAW) – значение REF, указывающее на удаляемую строку.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Удаление объектов по ссылке REF.
- Тестирование корректности разрешения REF в физическую строку.
- Управление объектами без явного указания первичного ключа.

Примеры запросов:

```
TRUNCATE utl_ref_demo;
INSERT INTO utl_ref_demo (id, payload) VALUES (1, 'to delete');

SELECT utl_ref_delete_object(pg_temp.utl_make_ref(1));

SELECT count(*) AS remaining_rows
FROM utl_ref_demo
WHERE id = 1;
```

UTL_REF.LOCK_OBJECT

Процедура блокирует строку по значению REF.

Входящие параметры:

- ref (RAW) – значение REF, указывающее на блокируемую строку.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Установка блокировки на объект по REF.
- Контроль конкурентного доступа к строке.
- Проверка корректности механизма блокировки по ссылке.

Примеры запросов:

```
TRUNCATE utl_ref_demo;
INSERT INTO utl_ref_demo (id, payload) VALUES (1, 'to lock');

BEGIN;
SELECT utl_ref_lock_object(pg_temp.utl_make_ref(1));
SELECT 'row with id = 1 locked in current transaction' AS result;
ROLLBACK;
```

UTL_REF.LOCK_OBJECT_OBJ

Функция блокирует строку по значению REF и возвращает объект.

Входящие параметры:

- ref (RAW) – значение REF, указывающее на блокируемую строку.

Выходные параметры:

- Возвращает представление объекта, соответствующего строке.

Типичные сценарии использования:

- Одновременная блокировка и чтение объекта по REF.
- Получение содержимого строки в защищённом режиме.
- Тестирование возврата объекта при блокировке.

Примеры запросов:

```
TRUNCATE utl_ref_demo;
INSERT INTO utl_ref_demo (id, payload) VALUES (1, 'locked object');

BEGIN;
SELECT convert_from(t.object::bytea, 'UTF8') AS result
FROM utl_ref_lock_object_obj(pg_temp.utl_make_ref(1)) AS t(object blob);
ROLLBACK;
```

UTL_REF.SELECT_OBJECT

Функция возвращает объект по значению REF.

Входящие параметры:

- ref (RAW) – значение REF, указывающее на строку.

Выходные параметры:

- Возвращает представление объекта, соответствующего строке.

Типичные сценарии использования:

- Получение объекта по ссылке REF.
- Чтение данных без явного знания структуры ключа.
- Тестирование разрешения ссылки в содержимое строки.

Примеры запросов:

```
TRUNCATE utl_ref_demo;
INSERT INTO utl_ref_demo (id, payload) VALUES (1, 'selected object');

SELECT convert_from(t.object::bytea, 'UTF8') AS result
FROM utl_ref_select_object(pg_temp.utl_make_ref(1)) AS t(object blob);
```

UTL_REF.UPDATE_OBJECT

Процедура обновляет строку по значению REF на основе переданного представления объекта.

Входящие параметры:

- ref (RAW) – значение REF, указывающее на обновляемую строку.
- object (BLOB) – новое представление объекта.

Выходные параметры:

- Нет.

Типичные сценарии использования:

- Обновление объекта по ссылке REF.
- Передача нового содержимого строки в сериализованном виде.
- Тестирование записи по объектной ссылке.

Примеры запросов:

```
TRUNCATE utl_ref_demo;
INSERT INTO utl_ref_demo (id, payload) VALUES (1, 'before update');

SELECT utl_ref_update_object(
    pg_temp.utl_make_ref(1),
    convert_to('{"id":1,"payload":"after update"}', 'UTF8')::blob
);

SELECT row_to_json(t)::text AS result
FROM (
    SELECT id, payload
    FROM utl_ref_demo
    WHERE id = 1
) AS t;
```

UTL_REF_UPDATE_OBJ_V2

Вспомогательная функция для: utl_ref_update_object

Описание: обновляет объект по REF-ссылке с заданным режимом записи.

Входящие параметры:

- ref (RAW) – REF-ссылка на объект
- object (BLOB) – новые данные объекта
- write_mode (INT4) – режим записи

Выходные параметры:

- VOID – функция не возвращает значения

Типичные сценарии использования:

- Обновление объектов по ссылке
- Работа с объектными типами

Примеры запросов:

```
SELECT utl_ref_update_obj_v2(ref_raw, obj_blob, 1);
```

UTL_REF_GET_VERSION

Функция возвращает номер версии пакета UTL_REF для работы с REF-ссылками на объекты.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для объектных ссылок
- Совместимость навигации

Примеры запросов:

```
SELECT utl_ref_get_version();
```

5.2.51. Пакет UTL_GDK

UTL_GDK_GET_VERSION

Возвращает версию реализации UTL_GDK.

Входящие параметры:

- Отсутствуют.

Выходные параметры:

- TEXT - номер версии пакета.

Типичные сценарии использования:

- Получение информации о версии пакета.

Примеры запросов:

```
SELECT utl_gdk_get_version() AS package_version;
```

UTL_GDK_ORACLE_CHARSET_TO_IANA

Преобразует Oracle charset name в IANA charset name.

Входящие параметры:

- oracle_charset TEXT - Oracle-имя набора символов

Выходные параметры:

- TEXT - IANA-имя кодировки либо SQL `NULL`.

Типичные сценарии использования:

- Получение IANA-кодировки.

Примеры запросов:

```
SELECT
    'AL32UTF8' AS oracle_charset,
    utl_gdk_oracle_charset_to_iana('AL32UTF8') AS iana_charset;
```

UTL_GDK_CHARSET_MAP

Отображает имя кодировки Oracle в IANA-имя кодировки.

Входящие параметры:

- TEXT - Oracle-имя набора символов

Выходные параметры:

- TEXT - IANA-имя кодировки либо SQL `NULL`.

Типичные сценарии использования:

- Получение IANA-кодировки.

Примеры запросов:

```
SELECT
    'AL32UTF8' AS input_charset,
    utl_gdk_charset_map('AL32UTF8') AS mapped_charset;
```

UTL_GDK_CHARSET_MAP(TEXT, INT4)

Отображает имя кодировки в заданном направлении Oracle/IANA.

Входящие параметры:

- TEXT - Oracle-имя набора символов.
- INT4 - Флаг направления.

Выходные параметры:

- TEXT - строка с найденным именем кодировки либо SQL NULL.

Типичные сценарии использования:

- Конвертация кодировки.

Примеры запросов:

```
SELECT
    'UTF-8' AS input_charset,
    2 AS map_flag,
    utl_gdk_charset_map('UTF-8', 2) AS mapped_charset;
```

UTL_GDK_LANGUAGE_MAP

Отображает имя языка Oracle в ISO-код языка.

Входящие параметры:

- TEXT - Oracle-имя набора символов.

Выходные параметры:

- TEXT - строка с ISO-кодом языка либо SQL NULL.

Типичные сценарии использования:

- Конвертация кодировки.

Примеры запросов:

```
SELECT
    'AMERICAN' AS oracle_language,
    utl_gdk_language_map('AMERICAN') AS mapped_language;
```

UTL_GDK_LANGUAGE_MAP(TEXT, INT4)

Отображает имя языка в заданном направлении Oracle/ISO. flag=4 используется для обратного направления: ISO language → Oracle language.

Входящие параметры:

- TEXT - Oracle-имя набора символов.
- INT4 - Флаг направления.

Выходные параметры:

- TEXT - строка с найденным именем языка либо SQL NULL.

Типичные сценарии использования:

- Конвертация кодировки.

Примеры запросов:

```
SELECT
    'fr' AS input_language,
    4 AS map_flag,
    utl_gdk_language_map('fr', 4) AS mapped_language;
```

UTL_GDK_TERRITORY_MAP

Отображает имя территории Oracle в ISO 3166-1 alpha-2

Входящие параметры:

- TEXT - Oracle-имя территории.

Выходные параметры:

- TEXT - строка с ISO alpha-2 кодом территории либо SQL NULL.

Типичные сценарии использования:

- Конвертация кодов территорий.

Примеры запросов:

```
SELECT
    'AMERICA' AS oracle_territory,
    utl_gdk_territory_map('AMERICA') AS mapped_territory;
```

UTL_GDK_TERRITORY_MAP(TEXT, INT4)

Отображает имя территории в заданном направлении Oracle/ISO

Входящие параметры:

- TEXT - имя территории.
- INT4 - Флаг направления.

Выходные параметры:

- TEXT - строка с найденным именем/кодом территории либо SQL NULL.

Типичные сценарии использования:

- Конвертация кодов территорий.

Примеры запросов:

```
SELECT
    'USA' AS input_territory,
    6 AS map_flag,
    utl_gdk_territory_map('USA', 6) AS mapped_territory;
```

5.2.52. Пакет PLVDATE

PLVDATE.ADD_BIZDAYS

Функция добавляет указанное количество рабочих дней к заданной дате, исключая выходные (субботу и воскресенье).

Основные возможности:

- добавляет рабочие дни к указанной дате

- исключает выходные (субботу и воскресенье) из расчета
- поддерживает как положительные, так и отрицательные значения дней
- не учитывает праздничные дни (только стандартные выходные)

Входные параметры:

- base_date - базовая дата (тип DATE)
- days_to_add - количество рабочих дней для добавления (тип INTEGER)

Выходные параметры:

DATE - результирующая дата после добавления рабочих дней:

- Будущая дата, если days_to_add > 0
- Прошедшая дата, если days_to_add < 0
- Исходная дата, если days_to_add = 0

Типичные сценарии использования:

- расчет даты выполнения задачи с учетом только рабочих дней
- определение сроков оплаты по договорам
- планирование сроков поставки товаров
- вычисление дат для бизнес-процессов

Примеры запросов:

```
select plvdate_add_bizdays(date '2025-10-08', 1);
```

PLVDATE.BIZDAYS_BETWEEN

Функция вычисляет количество рабочих дней между двумя датами, исключая выходные (субботу и воскресенье).

Основные возможности:

- Подсчитывает количество рабочих дней между двумя датами
- Исключает выходные (субботу и воскресенье) из расчета
- Возвращает положительное или отрицательное значение в зависимости от

порядка дат

- Не учитывает праздничные дни (только стандартные выходные)

Входные параметры:

- start_date - начальная дата (тип DATE)
- end_date - конечная дата (тип DATE)

Выходные параметры:

INTEGER - количество рабочих дней между датами: * Положительное число, если end_date > start_date * Отрицательное число, если end_date < start_date * 0, если даты равны

Типичные сценарии использования:

- Расчет длительности рабочих процессов
- Определение количества рабочих дней для оплаты
- Анализ сроков выполнения задач
- Планирование ресурсов на основе рабочих дней

Примеры запросов:

```
select plvdate_bizdays_between(date '2025-10-08', date '2025-10-10');
```

PLVDATE.INCLUDE_START

Функция определяет, следует ли включать начальную дату при расчете временных интервалов.

Основные возможности:

- Управляет включением начальной даты в расчет интервалов
- Используется совместно с другими функциями работы с датами
- Поддерживает различные режимы расчета временных периодов

Входные параметры:

include_flag (BOOLEAN) - флаг включения начальной даты: * TRUE - начальная дата включается в расчет * FALSE - начальная дата исключается из расчета

Выходные параметры:

- VOID - функция устанавливает параметр для текущей сессии

Типичные сценарии использования:

- Точный расчет временных интервалов
- Финансовые расчеты с учетом/без учета начальной даты
- Определение сроков выполнения задач
- Логика работы с периодами дат

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_days INTEGER;
BEGIN
    -- Устанавливаем режим "включать начальную дату"
    PERFORM PLVDATE_INCLUDE_START();

    -- Теперь считаем рабочие дни с учетом начальной даты
    v_days := PLVDATE_BIZDAYS_BETWEEN(DATE '2025-10-08', DATE '2025-10-10');
    RAISE NOTICE 'C INCLUDE_START: % рабочих дней', v_days;

    -- Устанавливаем режим "НЕ включать начальную дату"
    PERFORM PLVDATE_NOINCLUDE_START();

    -- Считаем рабочие дни БЕЗ учета начальной даты
    v_days := PLVDATE_BIZDAYS_BETWEEN(DATE '2025-10-08', DATE '2025-10-10');
    RAISE NOTICE 'C NOINCLUDE_START: % рабочих дней', v_days;

    -- Проверяем текущий режим через INCLUDING_START (должен вернуть TRUE/FALSE)
    PERFORM PLVDATE_INCLUDING_START();
END;
$$;
```

PLVDATE.INCLUDING_START

Функция возвращает текущее значение флага, определяющего включение начальной даты в расчет временных интервалов.

Основные возможности:

- Позволяет получить текущее состояние флага включения начальной даты
- Используется для проверки настроек расчета временных периодов
- Работает в сочетании с PLVDATE.INCLUDE_START

Входные параметры:

- Нет входных параметров

Выходные параметры:

BOOLEAN - текущее значение флага: * TRUE - начальная дата включается в расчет * FALSE - начальная дата исключается из расчета

Типичные сценарии использования:

- Проверка текущих настроек расчета дат
- Отладка расчетов временных интервалов
- Согласование поведения разных функций работы с датами
- Логирование параметров временных расчетов

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_date_1 DATE := DATE '2025-10-08';
    v_date_2 DATE := DATE '2025-10-10';
    v_result INTEGER;
    v_is_start_including BOOLEAN;
BEGIN
    -- Тест с INCLUDING START
    PERFORM PLVDATE_INCLUDE_START();
    v_result := PLVDATE_BIZDAYS_BETWEEN(v_date_1, v_date_2);
    v_is_start_including := PLVDATE_INCLUDING_START();

    -- ВЫВОДИМ результат проверки режима
    PERFORM DBMS_OUTPUT_PUT_LINE('Режим INCLUDING_START: ' || v_is_start_including);
    PERFORM DBMS_OUTPUT_PUT_LINE('Включая стартовый день: ' || v_result);

    -- Тест с NOINCLUDE START
    PERFORM PLVDATE_NOINCLUDE_START();
    v_is_start_including := PLVDATE_INCLUDING_START();
    v_result := PLVDATE_BIZDAYS_BETWEEN(v_date_1, v_date_2);

    -- ВЫВОДИМ результат проверки режима
    PERFORM DBMS_OUTPUT_PUT_LINE('Режим INCLUDING_START: ' || v_is_start_including);
    PERFORM DBMS_OUTPUT_PUT_LINE('Не включая стартовый день: ' || v_result);
END;
$$;
```

PLVDATE.ISBIZDAY

Функция проверяет, является ли указанная дата рабочим днём (учитывая только стандартные выходные - субботу и воскресенье).

Основные возможности:

- Проверяет, является ли дата рабочим днём
- Автоматически определяет выходные (суббота и воскресенье)
- Не учитывает праздничные дни (только стандартные выходные)
- Возвращает булево значение (TRUE/FALSE)

Входные параметры:

- check_date - проверяемая дата (тип DATE)

Выходные параметры:

BOOLEAN: * TRUE - если дата рабочий день (понедельник-пятница) * FALSE - если дата выходной (суббота или воскресенье)

Типичные сценарии использования:

- Проверка дат перед планированием задач
- Валидация рабочих дней для процессов
- Фильтрация данных по рабочим дням
- Интеграция с другими функциями работы с датами

Примеры запросов:

```
select plvdate_default_holidays('RUSSIA');
select plvdate_isbizday(date '2025-01-01');
```

PLVDATE.ISLEAPYEAR

Функция проверяет, является ли указанный год високосным.

Основные возможности:

- Определяет високосность года по григорианскому календарю
- Работает с годами в широком диапазоне
- Возвращает булево значение (TRUE/FALSE)
- Не требует дополнительных параметров

Входные параметры:

- year - проверяемый год (тип INTEGER)

Выходные параметры:

BOOLEAN: * TRUE - если год високосный * FALSE - если год не високосный

Типичные сценарии использования:

- Проверка високосности года для календарных расчетов

- Определение количества дней в феврале
- Валидация дат в информационных системах
- Финансовые расчеты с учетом длины года

Примеры запросов:

```
select plvdate_isleapyear(date '2024-01-01');
select plvdate_isleapyear(date '2023-01-01');
```

PLVDATE.NEXT_BIZDAY

Функция возвращает следующий рабочий день после указанной даты (исключая выходные).

Основные возможности:

- Находит следующий рабочий день (понедельник-пятница)
- Если входная дата уже рабочий день, возвращает следующий рабочий день
- Не учитывает праздничные дни (только стандартные выходные)
- Всегда возвращает дату в будущем относительно входной даты

Входные параметры:

- input_date - исходная дата (тип DATE)

Выходные параметры:

- DATE - следующий рабочий день после указанной даты

Типичные сценарии использования:

- Расчет сроков выполнения задач
- Определение дат начала процессов
- Планирование последующих рабочих дней
- Автоматизация рабочих календарей

Примеры запросов:

```
select plvdate_next_bizday(date '2024-11-09');
```

PLVDATE.NOINCLUDE_START

Функция отключает включение начальной даты в расчет временных интервалов для текущей сессии.

Основные возможности:

- Устанавливает флаг исключения начальной даты из расчетов
- Влияет на функции работы с временными интервалами
- Действует в рамках текущей сессии
- Является антагонистом plvdate_include_start(FALSE)

Входные параметры:

- Нет входных параметров

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Настройка расчета интервалов без учета начальной даты
- Финансовые расчеты периодов
- Определение сроков выполнения задач
- Синхронизация с бизнес-логикой приложений

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_date_1 DATE := DATE '2025-10-08';
    v_date_2 DATE := DATE '2025-10-10';
    v_result INTEGER;
    v_is_start_including BOOLEAN;
BEGIN
    -- Тест с INCLUDE_START (включаем стартовый день)
    PERFORM PLVDATE_INCLUDE_START();
    v_result := PLVDATE_BIZDAYS_BETWEEN(v_date_1, v_date_2);
    v_is_start_including := PLVDATE_INCLUDING_START();

    IF v_is_start_including THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Включая стартовый день: ' || v_result);
    END IF;

    -- Тест с NOINCLUDE_START (НЕ включаем стартовый день)
    PERFORM PLVDATE_NOINCLUDE_START();
    v_is_start_including := PLVDATE_INCLUDING_START();
    v_result := PLVDATE_BIZDAYS_BETWEEN(v_date_1, v_date_2);

    IF NOT v_is_start_including THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Не включая стартовый день: ' || v_result);
    END IF;
END;
$$;
```

PLVDATE.PREV_BIZDAY

Функция возвращает предыдущий рабочий день перед указанной датой (исключая выходные).

Основные возможности:

- Находит предыдущий рабочий день (понедельник-пятница)
- Если входная дата уже рабочий день, возвращает предыдущий рабочий день
- Не учитывает праздничные дни (только стандартные выходные)
- Всегда возвращает дату в прошлом относительно входной даты

Входные параметры:

- input_date - исходная дата (тип DATE)

Выходные параметры:

- DATE - предыдущий рабочий день перед указанной датой

Типичные сценарии использования:

- Определение последнего рабочего дня перед событием
- Расчет сроков выполнения задач в обратном порядке
- Анализ рабочих периодов в прошлом
- Корректировка отчетных дат на рабочие дни

Примеры запросов:

```
select plvdate_prev_bizday(date '2024-11-05');
```

PLVDATE.SET_NONBIZDAY

Функция позволяет установить указанную дату как нерабочий день (выходной или праздничный) в рамках текущей сессии.

Основные возможности:

- Добавляет пользовательские нерабочие дни в текущую сессию
- Поддерживает различные форматы ввода дат
- Позволяет устанавливать дни недели как регулярные выходные
- Влияет на расчеты всех функций работы с рабочими днями

Входные параметры:

- date_expr - дата или день недели для установки как нерабочий (тип DATE/TEXT)
- is_recurring - необязательный флаг регулярности (только для дней недели)

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Объявление разовых праздничных дней
- Установка регулярных выходных дней
- Настройка календаря для тестирования
- Учет локальных нерабочих дней

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_date DATE := DATE '2025-10-09';
    v_is_bizday BOOLEAN;
BEGIN
    -- Проверяем исходный статус (должен быть рабочим днём, если не праздник)
    v_is_bizday := PLVDATE_ISBIZDAY(v_date);
    PERFORM DBMS_OUTPUT_PUT_LINE('1. Исходный статус 2025-10-09: ' || v_is_bizday);

    -- Устанавливаем как НЕ рабочий день
    PERFORM PLVDATE_SET_NONBIZDAY(v_date);
    PERFORM DBMS_OUTPUT_PUT_LINE('2. Установили NONBIZDAY для 2025-10-09');

    -- Проверяем после установки (должен быть FALSE)
    v_is_bizday := PLVDATE_ISBIZDAY(v_date);
    PERFORM DBMS_OUTPUT_PUT_LINE('3. Статус после SET_NONBIZDAY: ' || v_is_bizday);

    -- Отменяем установку (возвращаем как рабочий день)
    PERFORM PLVDATE_UNSET_NONBIZDAY(v_date);
    PERFORM DBMS_OUTPUT_PUT_LINE('4. Отменили NONBIZDAY для 2025-10-09');

    -- Проверяем после отмены (должен вернуться в исходный статус)
    v_is_bizday := PLVDATE_ISBIZDAY(v_date);
    PERFORM DBMS_OUTPUT_PUT_LINE('5. Статус после UNSET_NONBIZDAY: ' || v_is_bizday);
END;
$$;
```

PLVDATE.UNSET_NONBIZDAY

Функция отменяет установку указанной даты или дня недели как нерабочего дня в рамках текущей сессии.

Основные возможности:

- Отменяет действие PLVDATE.SET_NONBIZDAY для указанных дат
- Поддерживает те же форматы ввода, что и SET_NONBIZDAY
- Позволяет отменять как разовые, так и регулярные нерабочие дни
- Восстанавливает стандартное поведение функций расчета рабочих дней

Входные параметры:

- date_expr - дата или день недели для отмены нерабочего статуса (тип DATE/TEXT)
- is_recurring - необязательный флаг регулярности (только для дней недели)

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Отмена временных праздничных дней
- Возврат стандартных выходных дней в рабочие
- Корректировка рабочего календаря
- Тестирование альтернативных сценариев

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_date DATE := DATE '2025-10-09';
    v_is_bizday BOOLEAN;
BEGIN
    -- Проверяем исходный статус
    v_is_bizday := PLVDATE_ISBIZDAY(v_date);
    IF v_is_bizday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Указанная дата: ' || v_date || ' - является
рабочим днем');
    END IF;

    -- Устанавливаем как НЕ рабочий день
    PERFORM DBMS_OUTPUT_PUT_LINE('Задаем указанную дату: ' || v_date || ' - как НЕ
рабочий день');
    PERFORM PLVDATE_SET_NONBIZDAY(v_date);

    -- Проверяем после установки
    v_is_bizday := PLVDATE_ISBIZDAY(v_date);
    IF NOT v_is_bizday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Указанная дата: ' || v_date || ' - теперь НЕ
является рабочим днем');
    END IF;

    -- Отменяем установку (тестируем UNSET_NONBIZDAY)
    PERFORM DBMS_OUTPUT_PUT_LINE('Задаем дату: ' || v_date || ' - снова как рабочий
день');
    PERFORM PLVDATE_UNSET_NONBIZDAY(v_date);

    -- Проверяем после отмены
    v_is_bizday := PLVDATE_ISBIZDAY(v_date);
    IF v_is_bizday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Указанная дата: ' || v_date || ' - снова
является рабочим днем');
    END IF;
END;
$$;
```

PLVDATE.UNUSE_EASTER

Функция отключает учет Пасхи и пасхальных праздников в расчетах рабочих дней для текущей сессии.

Основные возможности:

- Отключает автоматический учет Пасхи и связанных праздников
- Влияет на все функции работы с рабочими днями
- Действует только в рамках текущей сессии
- Позволяет временно исключить пасхальные праздники из расчетов

Входные параметры:

- Нет входных параметров

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Временное отключение пасхальных праздников для тестирования
- Расчеты без учета плавающих религиозных праздников
- Сравнение результатов с разными календарными настройками
- Адаптация расчетов под специфические бизнес-правила

Примеры запросов:

```
\serveroutput on

DO $$
BEGIN
    -- Шаг 1: Включаем учет Пасхи
    PERFORM PLVDATE_USE_EASTER();
    PERFORM DBMS_OUTPUT_PUT_LINE('После USE_EASTER: ' || PLVDATE_USING_EASTER());

    -- Шаг 2: Отключаем учет Пасхи (тестируем UNUSE_EASTER)
    PERFORM PLVDATE_UNUSE_EASTER();
    PERFORM DBMS_OUTPUT_PUT_LINE('После UNUSE_EASTER: ' || PLVDATE_USING_EASTER());

    -- Шаг 3: Снова включаем (восстанавливаем исходное состояние)
    PERFORM PLVDATE_USE_EASTER();
    PERFORM DBMS_OUTPUT_PUT_LINE('После повторного USE_EASTER: ' ||
PLVDATE_USING_EASTER());
END;
$$;
```

PLVDATE.UNUSE_GREAT_FRIDAY

Функция отключает учет Страстной пятницы (Great Friday) в расчетах рабочих дней для текущей сессии.

Основные возможности:

- Отключает автоматический учет Страстной пятницы как праздничного дня
- Влияет на все функции работы с рабочими днями
- Действует только в рамках текущей сессии
- Позволяет временно исключить этот религиозный праздник из расчетов

Входные параметры:

- Нет входных параметров

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Временное отключение религиозных праздников для тестирования

- Расчеты без учета Страстной пятницы
- Сравнение результатов с разными календарными настройками
- Адаптация расчетов под нерелигиозные бизнес-правила

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_is_using_great_friday BOOLEAN;
BEGIN
    -- Шаг 1: Проверяем исходное состояние (должно быть TRUE, если включено по
умолчанию)
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Исходное состояние: Страстная пятница является
выходным днем');
    ELSE
        PERFORM DBMS_OUTPUT_PUT_LINE('Исходное состояние: Страстная пятница НЕ
является выходным днем');
    END IF;

    -- Шаг 2: Принудительно включаем Страстную пятницу (чтобы тест был наглядным)
    PERFORM PLVDATE_USE_GREAT_FRIDAY();
    PERFORM DBMS_OUTPUT_PUT_LINE('После USE_GREAT_FRIDAY: Страстная пятница
включена');
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Проверка: Страстная пятница является выходным
днем');
    END IF;

    -- Шаг 3: Отключаем Страстную пятницу (тестируем UNUSE_GREAT_FRIDAY)
    PERFORM PLVDATE_UNUSE_GREAT_FRIDAY();
    PERFORM DBMS_OUTPUT_PUT_LINE('После UNUSE_GREAT_FRIDAY: Страстная пятница
отключена');
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF NOT v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Проверка: Страстная пятница НЕ является выходным
днем');
    END IF;

    -- Шаг 4: Снова включаем (восстанавливаем)
    PERFORM PLVDATE_USE_GREAT_FRIDAY();
```

```
    PERFORM DBMS_OUTPUT_PUT_LINE('После повторного USE_GREAT_FRIDAY: Страстная пятница
снова включена');
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Проверка: Страстная пятница снова является
выходным днем');
    END IF;
END;
$$;
```

PLVDATE.USE_EASTER

Функция включает учет Пасхи и связанных пасхальных праздников в расчетах рабочих дней для текущей сессии.

Основные возможности:

- Включает автоматический учет Пасхи и пасхальных праздников
- Влияет на все функции работы с рабочими днями
- Действует только в рамках текущей сессии
- Позволяет временно учитывать плавающие религиозные праздники

Входные параметры:

- Нет входных параметров

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Включение пасхальных праздников для расчетов
- Тестирование функций с учетом религиозных праздников
- Сравнение результатов с разными календарными настройками
- Адаптация расчетов под региональные праздники

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_is_using_easter BOOLEAN;
BEGIN
    -- Шаг 1: Отключаем Пасху (чтобы потом тестировать включение)
    PERFORM PLVDATE_UNUSE_EASTER();
    PERFORM DBMS_OUTPUT_PUT_LINE('После UNUSE_EASTER: Пасха отключена');
    v_is_using_easter := PLVDATE_USING_EASTER();
    IF NOT v_is_using_easter THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Пасха НЕ является выходным днем');
    END IF;

    -- Шаг 2: Включаем Пасху (тестируем USE_EASTER)
    PERFORM PLVDATE_USE_EASTER();
    PERFORM DBMS_OUTPUT_PUT_LINE('После USE_EASTER: Пасха включена');
    v_is_using_easter := PLVDATE_USING_EASTER();
    IF v_is_using_easter THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Пасха является выходным днем');
    END IF;

    -- Шаг 3: Снова отключаем (восстанавливаем исходное состояние)
    PERFORM PLVDATE_UNUSE_EASTER();
    PERFORM DBMS_OUTPUT_PUT_LINE('После повторного UNUSE_EASTER: Пасха снова
отключена');
    v_is_using_easter := PLVDATE_USING_EASTER();
    IF NOT v_is_using_easter THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Пасха снова НЕ является выходным днем');
    END IF;
END;
$$;
```

PLVDATE.USE_GREAT_FRIDAY

Функция включает учет Страстной пятницы (Great Friday) в расчетах рабочих дней для текущей сессии.

Основные возможности:

- Включает автоматический учет Страстной пятницы как праздничного дня
- Влияет на все функции работы с рабочими днями
- Действует только в рамках текущей сессии

- Позволяет временно учитывать этот религиозный праздник

Входные параметры:

- Нет входных параметров

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Включение Страстной пятницы для расчетов
- Тестирование функций с учетом религиозных праздников
- Сравнение результатов с разными календарными настройками
- Адаптация расчетов под региональные праздники

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_is_using_great_friday BOOLEAN;
BEGIN
    -- Шаг 1: Отключаем Страстную пятницу (чтобы потом тестировать включение)
    PERFORM PLVDATE_UNUSE_GREAT_FRIDAY();
    PERFORM DBMS_OUTPUT_PUT_LINE('После UNUSE_GREAT_FRIDAY: Страстная пятница
отключена');
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF NOT v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Страстная пятница НЕ является выходным днем');
    END IF;

    -- Шаг 2: Включаем Страстную пятницу (тестируем USE_GREAT_FRIDAY)
    PERFORM PLVDATE_USE_GREAT_FRIDAY();
    PERFORM DBMS_OUTPUT_PUT_LINE('После USE_GREAT_FRIDAY: Страстная пятница
включена');
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Страстная пятница является выходным днем');
    END IF;

    -- Шаг 3: Снова отключаем (восстанавливаем исходное состояние)
    PERFORM PLVDATE_UNUSE_GREAT_FRIDAY();
    PERFORM DBMS_OUTPUT_PUT_LINE('После повторного UNUSE_GREAT_FRIDAY: Страстная
пятница снова отключена');
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF NOT v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Страстная пятница снова НЕ является выходным
днем');
    END IF;
END;
$$;
```

PLVDATE.USING_EASTER

Функция проверяет, учитывается ли Пасха и пасхальные праздники в текущей сессии.

Основные возможности:

- Позволяет проверить текущее состояние учета Пасхи
- Возвращает булево значение (TRUE/FALSE)

- Используется для отладки и проверки настроек

Входные параметры:

- Нет входных параметров

Выходные параметры:

BOOLEAN: * TRUE - если учет Пасхи включен * FALSE - если учет Пасхи выключен

Типичные сценарии использования:

- Проверка текущих настроек расчета рабочих дней
- Отладка функций работы с датами
- Условное выполнение операций в зависимости от настроек
- Логирование параметров системы

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_is_using_easter BOOLEAN;
BEGIN
    -- Шаг 1: Проверяем текущее состояние USING_EASTER
    v_is_using_easter := PLVDATE_USING_EASTER();
    IF v_is_using_easter THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Пасха является выходным днем');
    ELSE
        PERFORM DBMS_OUTPUT_PUT_LINE('Пасха НЕ является выходным днем');
    END IF;

    -- Шаг 2: Меняем состояние (отключаем)
    PERFORM PLVDATE_UNUSE_EASTER();
    v_is_using_easter := PLVDATE_USING_EASTER();
    IF NOT v_is_using_easter THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('После UNUSE: Пасха НЕ является выходным днем');
    END IF;

    -- Шаг 3: Меняем состояние обратно (включаем)
    PERFORM PLVDATE_USE_EASTER();
    v_is_using_easter := PLVDATE_USING_EASTER();
    IF v_is_using_easter THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('После USE: Пасха снова является выходным днем');
    END IF;
END;
$$;
```

PLVDATE.USING_GREAT_FRIDAY

Функция проверяет, учитывается ли Страстная пятница (Great Friday) в расчетах рабочих дней для текущей сессии.

Основные возможности:

- Позволяет проверить текущее состояние учета Страстной пятницы
- Возвращает булево значение (TRUE/FALSE)
- Используется для отладки и проверки настроек

Входные параметры:

- Нет входных параметров

Выходные параметры:

BOOLEAN: * TRUE - если учет Страстной пятницы включен * FALSE - если учет Страстной пятницы выключен

Типичные сценарии использования:

- Проверка текущих настроек праздничных дней
- Отладка расчетов рабочих дней
- Условная логика в зависимости от настроек
- Валидация конфигурации системы

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_is_using_great_friday BOOLEAN;
BEGIN
    -- Шаг 1: Проверяем текущее состояние USING_GREAT_FRIDAY
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('Страстная пятница является выходным днем');
    ELSE
        PERFORM DBMS_OUTPUT_PUT_LINE('Страстная пятница НЕ является выходным днем');
    END IF;

    -- Шаг 2: Меняем состояние (отключаем)
    PERFORM PLVDATE_UNUSE_GREAT_FRIDAY();
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF NOT v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('После UNUSE: Страстная пятница НЕ является
выходным днем');
    END IF;

    -- Шаг 3: Меняем состояние обратно (включаем)
    PERFORM PLVDATE_USE_GREAT_FRIDAY();
    v_is_using_great_friday := PLVDATE_USING_GREAT_FRIDAY();
    IF v_is_using_great_friday THEN
        PERFORM DBMS_OUTPUT_PUT_LINE('После USE: Страстная пятница снова является
выходным днем');
    END IF;
END;
$$;
```

PLVDATE_INCLUDE_ANCHOR

Вспомогательная функция для: plvdate_include_start, plvdate_noinclude_start

Описание: задаёт режим включения начальной даты в расчёты.

Входящие параметры:

- include_flag (BOOLEAN) – флаг включения начальной даты
- anchor_mode (INT4) – режим якорной даты

Выходные параметры:

- VOID – функция не возвращает значения

Типичные сценарии использования:

- Настройка границ интервалов дат
- Расчет периодов с включением или исключением границ

Примеры запросов:

```
PERFORM plvdate_include_anchor(TRUE, 1);
```

PLVDATE_GET_VERSION

Функция возвращает номер версии пакета PLVDATE для работы с датами.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для операций с датами
- Совместимость форматов

Примеры запросов:

```
SELECT plvdate_get_version();
```

5.2.53. Пакет PLVSTR

PLVSTR.BETWN

Функция извлекает подстроку между указанными разделителями из исходной строки.

Основные возможности:

- Извлекает часть строки между двумя указанными разделителями

- Поддерживает многократное вхождение разделителей
- Позволяет задавать начальную позицию поиска
- Возвращает NULL, если разделители не найдены

Входные параметры:

- string_in - исходная строка (TEXT)
- start_delim - начальный разделитель (TEXT)
- end_delim - конечный разделитель (TEXT)
- start_pos - необязательная начальная позиция поиска (INTEGER, по умолчанию 1)
- nth_appearance - необязательный номер вхождения (INTEGER, по умолчанию 1)

Выходные параметры:

- TEXT - найденная подстрока или NULL

Типичные сценарии использования:

- Парсинг структурированных текстовых данных
- Извлечение значений из строк с разделителями
- Обработка логов и системных сообщений
- Анализ текстовых шаблонов

Примеры запросов:

```
select plvstr_betwn('user@sub.domain.example.com', '@', '.', 1, 1);
```

PLVSTR.SWAP

Функция возвращает результат замены подстроки в исходной строке.

Входящие параметры:

- str – исходная строка
- substr – символы для подстановки
- start – начальный индекс подстроки в исходной строке

- len – количество заменяемых символов

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Быстрое редактирование строк с заменой определенных символов.
- Корректировка опечаток в больших текстах или документах.
- Преобразование данных для удобства дальнейшей обработки.

Примеры запросов:

```
select plvstr_swap('2023-12-31', '2024', 1, 4);
```

PLVSTR.RIGHT

Функция возвращает указанное количество символов с конца строки.

Входящие параметры:

- str – исходная строка
- num – количество символов, которое будет включено с правой стороны, либо исключено из исходной строки с левой стороны (при отрицательном значении)

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Получение расширения файла из пути.
- Извлечение последних символов из почтовых адресов или идентификаторов.
- Нормализация строк, удаление ненужных начальных символов.

Примеры запросов:

```
select plvstr_right('1234567812345678', 4);
```

PLVSTR.LEFT

Функция возвращает указанное количество символов с начала строки.

Входящие параметры:

- str - исходная строка
- num – количество символов, которое будет возвращено с левой стороны, либо исключено из исходной строки с правой стороны (при отрицательном значении)

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Извлечение первых символов почтового адреса или телефона.
- Нормализация строк, удаление лишних символов в конце.
- Получение префикса файла из пути.

Примеры запросов:

```
select plvstr_left('OpenAI', 4);
```

PLVSTR.RVRS

Функция возвращает реверсивное представление строки с возможностью уточнения диапазона индексов.

Входные параметры:

- str - исходная строка;
- start - начальный индекс исходной строки;
- end - конечный индекс исходной строки.

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Проверка палиндромов (слов, читающихся одинаково вперед и назад).
- Обратная сортировка текста для особых требований (например, криптография).
- Тестирование алгоритмов обработки обратных строк.

Пример запросов:

```
select plvstr_rvrs('Привет', 1);
```

PLVSTR.RSTRIP

Функция удаляет указанное количество вхождений подстроки с конца исходной строки. Если количество вхождений меньше 1 или подстрока отсутствует, возвращается исходная строка.

Входящие параметры:

- str — исходная строка; substr — подстрока для удаления; num — количество вхождений подстроки для удаления.

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Удаление суффиксов в наименованиях файлов.
- Обработка путей и ссылок для устранения лишних сегментов.
- Нормализация данных с повторяющимися частями в конце строки.

Примеры запросов:

```
select plvstr_rstrip('database.backup.bak', '.bak', 1);
```

PLVSTR.LSTRIP

Функция удаляет указанное количество вхождений подстроки с начала исходной строки. Если количество вхождений меньше 1 или подстрока отсутствует, возвращается исходная строка.

Входные параметры:

- str — исходная строка;
- substr — подстрока для удаления;
- num — количество вхождений подстроки для удаления.

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Удаление префиксов в наименованиях файлов или путях.
- Нормализация текстовых данных с повторяющимися частями в начале строки.
- Удаление ведущих символов или служебных элементов.

Примеры запросов:

```
select plvstr_lstrip('https://example.com', 'https://', 1);
```

PLVSTR.RPART

Функция возвращает подстроку, содержащую часть строки до указанного вхождения разделительного символа от конца строки. Возвращает NULL или исходную строку, если подстрока не найдена, в зависимости от параметра all_if_notfound.

Входящие параметры:

- str — исходная строка;
- div — подстрока для поиска;
- start — начальный индекс поиска;
- nth — номер вхождения подстроки для поиска;
- all_if_notfound — логическое значение, указывающее, нужно ли возвращать

исходную строку, если div не найден.

Выходящие параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Извлечение домена из URL.
- Отделение имен папок из путей файлов.
- Разделение сложного текста на составляющие по разделителям.

Примеры запросов:

```
select plvstr_rpart('document.backup.txt', '.', 1);
```

PLVSTR.LPART

Функция возвращает подстроку, содержащую часть строки до указанного вхождения разделительного символа от начала строки. Возвращает NULL или исходную строку, если подстрока не найдена, в зависимости от параметра all_if_notfound.

Входящие параметры:

- str — исходная строка;
- div — подстрока для поиска;
- start — начальный индекс поиска;
- nth — номер вхождения подстроки для поиска;
- all_if_notfound — логическое значение, указывающее, нужно ли возвращать исходную строку, если div не найден.

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Извлечение имени пользователя из email-адреса.
- Обработка иерархических путей и разделение их на компоненты.

- Парсинг и нормализация текстовых данных по известным разделителям.

Примеры запросов:

```
SELECT plvstr_lpart('C:\work\docs\file.txt', '\', 1, 1);
```

PLVSTR.INSTR

Функция возвращает индекс начала вхождения подстроки в строку. Возвращает 0, если подстрока отсутствует.

Входные параметры:

- str — исходная строка;
- substr — подстрока для поиска;
- start — начальный индекс поиска;
- nth — номер вхождения для поиска.

Выходные параметры:

- Функция возвращает значение типа int.

Типичные сценарии использования:

- Извлечение имени пользователя из email-адреса.
- Обработка иерархических путей и разделение их на компоненты.
- Парсинг и нормализация текстовых данных по известным разделителям.

Примеры запросов:

```
select plvstr_instr('банан', 'ан');
```

```
\serveroutput on

DO $$
DECLARE
    v_string TEXT := 'C:\work\docs\file.txt';
    v_substring TEXT := '\';
    v_position INTEGER;
BEGIN
    -- Ищем первое вхождение разделителя
    v_position := PLVSTR_INSTR(v_string, v_substring);
    PERFORM DBMS_OUTPUT_PUT_LINE('Позиция первого "' || v_substring || '": ' ||
v_position);

    -- Ищем с определенной позиции
    v_position := PLVSTR_INSTR(v_string, v_substring, 5);
    PERFORM DBMS_OUTPUT_PUT_LINE('Позиция после 5-го символа: ' || v_position);

    -- Ищем n-е вхождение
    v_position := PLVSTR_INSTR(v_string, v_substring, 1, 2);
    PERFORM DBMS_OUTPUT_PUT_LINE('Позиция 2-го вхождения: ' || v_position);

    v_position := PLVSTR_INSTR(v_string, v_substring, 1, 3);
    PERFORM DBMS_OUTPUT_PUT_LINE('Позиция 3-го вхождения: ' || v_position);

    v_position := PLVSTR_INSTR(v_string, v_substring, 1, 4);
    PERFORM DBMS_OUTPUT_PUT_LINE('Позиция 4-го вхождения (не существует): ' ||
v_position);
END;
$$;
```

```
select plvstr_instr('банан', 'а', 1, 2);
```

PLVSTR.SUBSTR

Функция возвращает подстроку строки text начиная с указанного индекса start. Дополнительно поддерживается указание длины подстроки len для ограничения количества символов. *Входные параметры:*

- str — строка, из которой извлекается подстрока;
- start — начальный индекс подстроки. Поддерживаются отрицательные значения для отсчета с конца строки;
- len — длина извлекаемой подстроки. Если значение < 1, возвращается NULL.

Выходные параметры:

- Функция возвращает значение типа text.

Типичные сценарии использования:

- Извлечение короткого сниппета из длинной строки.
- Обрезка лишнего текста для уменьшения объема данных.
- Стандартизация строк, убирая лишнюю информацию.

Примеры запросов:

```
select plvstr_substr('Привет', 1, 3);
select plvstr_substr('2024-12-31', 6, 2);
```

PLVSTR.IS_PREFIX

Функция проверяет, является ли строка prefix префиксом (началом) строки str. Дополнительно поддерживается указание чувствительности к регистру cs для управления точностью сравнения.

Входные параметры:

- str — исходная строка, начало которой проверяется;
- prefix — строка, наличие которой проверяется в начале строки str;
- cs — флаг чувствительности к регистру (true — регистр учитывается, false — не учитывается). Необязательный параметр, по умолчанию учитывает регистр.

Выходные параметры:

- Функция возвращает значение типа bool (true, если prefix является началом str, и false — в противном случае).

Типичные сценарии использования:

- Проверка соответствия файлового расширения (например, .txt, .pdf).
- Валидация вводимых данных (электронная почта, телефонный номер).
- Определение типов данных по префиксу (например, идентификаторы, коды продуктов).

- Маршрутизация или фильтрация данных на основе их начала.

Примеры запросов:

```
select plvstr_is_prefix('A', 'a', true);
select plvstr_is_prefix('a', 'a');
select plvstr_is_prefix('+79161234567', '+7');
```

Примечание: Функция также имеет перегрузки для работы с числовыми типами данных (int4, int8), которые преобразуют числа в строки и затем выполняют проверку префикса.

PLVSTR.NORMALIZE

Функция убирает все невидимые символы перед текстом и после.

Входящие параметры:

- str – входящая строка;

Выходящие параметры:

- Функция возвращает значение типа text. *Типичные сценарии использования:*
- Нормализация данных при импорте: очистка данных от лишних пробелов и непечатных символов.
- Подготовка данных для поиска: устранение лишних пробелов перед поиском.
- Обработка пользовательского ввода: очистка текста от лишних символов перед сохранением в базу данных.

Примеры запросов:

```
\serveroutput on

DO $$
DECLARE
    v_str VARCHAR := '     text      ';
    v_result VARCHAR;
BEGIN
    -- Выводим исходную строку с границами
    PERFORM DBMS_OUTPUT_PUT_LINE('Исходная строка: "' || v_str || '"');
    PERFORM DBMS_OUTPUT_PUT_LINE('Длина исходной строки: ' || LENGTH(v_str));

    -- Нормализуем строку (удаляем лишние пробелы)
    v_result := PLVSTR_NORMALIZE(v_str);

    -- Выводим результат с границами
    PERFORM DBMS_OUTPUT_PUT_LINE('После PLVSTR_NORMALIZE: "' || v_result || '"');
    PERFORM DBMS_OUTPUT_PUT_LINE('Длина результата: ' || LENGTH(v_result));
END;
$$;
```

```
select plvstr_normalize('   ул.    Полбина     дом      30      ');
```

PLVSTR_INSTR_SCAN

Вспомогательная функция для: plvstr_instr2, plvstr_instr3, plvstr_instr4

Описание: выполняет поиск подстроки с учетом параметров и профиля.

Входящие параметры:

- string_in (TEXT) – строка для поиска
- pattern (TEXT) – искомый образец
- start_in (INT4) – позиция начала поиска
- nth (INT4) – номер вхождения
- scan_profile (INT4) – профиль сканирования

Выходные параметры:

- INT4 – позиция найденного вхождения или 0

Типичные сценарии использования:

- Расширенный поиск подстрок
- Поиск с настраиваемыми параметрами

Примеры запросов:

```
SELECT plvstr_instr_scan('abracadabra','bra',1,2,0);
```

PLVSTR_GET_VERSION

Функция возвращает номер версии пакета PLVSTR для работы со строками.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для строковых операций
- Совместимость функций

Примеры запросов:

```
SELECT plvstr_get_version();
```

5.2.54. Пакет PLVCHR

PLVCHR.NTH

Функция возвращает символ строки, расположенный на указанной позиции.

Входящие параметры:

- str_in (VARCHAR2) – исходная строка, из которой необходимо получить символ.
- nth_in (INTEGER) – позиция символа в строке. Положительное значение задаёт позицию от начала строки, отрицательное — от конца строки.

Выходные параметры:

- Возвращает символ из указанной позиции строки.

Типичные сценарии использования:

- Получение символа по его позиции в строке.
- Извлечение символов с конца строки.
- Проверка содержимого строковых данных по фиксированным позициям.
- Использование в строковых преобразованиях и валидации данных.

Примеры запросов:

```
SELECT plvchr_nth('This is a very long string for testing purposes', 24); -- 'i' из
слова 'string'
SELECT plvchr_nth('Another long example string with many characters', -13); -- 'n' из
слова 'many'
```

PLVCHR_NTH_EXTRACT

Вспомогательная функция для: plvchr_nth

Описание: извлекает символ из строки по позиции с учетом профиля.

Входящие параметры:

- str_in (VARCHAR2) – исходная строка
- nth_in (INTEGER) – позиция символа (отрицательное значение – с конца)
- extract_profile (INT4) – профиль извлечения

Выходные параметры:

- TEXT – извлеченный символ

Типичные сценарии использования:

- Доступ к символам строки по индексу
- Обработка строк с разных направлений

Примеры запросов:

```
SELECT plvchr_nth_extract('example', -1, 1);
```

PLVCHR_GET_VERSION

Функция возвращает номер версии пакета PLVCHR для работы с символами.

Входящие параметры:

- Нет параметров

Выходные параметры:

- VARCHAR2 – строка с номером версии

Типичные сценарии использования:

- Проверка версии для символьных операций
- Совместимость функций

Примеры запросов:

```
SELECT plvchr_get_version();
```

6. In-memory-компонента

In-memory-компонента (DGrid) представляет собой внешнее оперативное хранилище данных, работающее отдельно от баз данных, сессий и транзакций. Оно используется для временного хранения значений, не связанных непосредственно с операциями внутри базы данных. Пакет DGrid позволяет пользователям сохранять (PUT_VALUE) и извлекать (GET_TEXT, GET_BYTEA) данные удобным способом, независимо от текущих состояний СУБД. Благодаря поддержке области видимости (scope), пользователи могут контролировать доступность сохраненных данных, делая их доступными либо глобально, либо ограниченно для конкретных пользователей.

Для работы доступны 3 области видимости:

- GLOBAL — значения остаются в общей памяти и видны всем пользователям без ограничений;
- SHARED — значения разделяются между всеми пользователями, но выделены в отдельную область от глобальных данных;
- USERSCOPE - значения доступны только текущему пользователю, сохраняются между его сессиями и недоступны другим пользователям.

Иное значение в области видимости (или его отсутствие) будет автоматически заменено на USERSCOPE.

Набор функций (диалекта PostgreSQL) для работы с in-memory-компонентой:

dgrid_put_value

Функция устанавливает значение для указанного ключа в заданном пространстве dgrid.

Основные возможности:

- Сохраняет текстовые (TEXT) и бинарные (BYTEA) значения
- Поддерживает различные пространства хранения (scopes)
- Позволяет организовать структурированное хранение данных
- Обеспечивает быстрый доступ к сохраненным значениям

Входные параметры:

- key - ключ для сохранения значения (тип TEXT)
- value - сохраняемое значение (тип TEXT или BYTEA)
- scope - необязательный параметр пространства хранения (тип dgrid_scope, по умолчанию USERSCOPE)

Выходные параметры:

- VOID - функция не возвращает значение

Типичные сценарии использования:

- Сохранение пользовательских настроек
- Кеширование данных приложения
- Обмен данными между сессиями
- Хранение бинарных объектов

Примеры запросов:

```
SELECT dgrid_put_value('test_key', 'test_value');
```

dgrid_get_text

Функция позволяет извлекать ранее установленное текстовое значение по ключу.

Основные возможности:

- Позволяет получать текстовые значения по ключу
- Поддерживает работу с различными пространствами хранения
- Возвращает NULL при отсутствии значения

Входные параметры:

- key (TEXT) - ключ для извлечения значения
- scope (dgrid_scope) - пространство хранения. По умолчанию: 'USERSCOPE'

Выходные параметры:

- TEXT - найденное текстовое значение

- NULL - если значение отсутствует

Типичные сценарии использования:

- Получение сохранённых текстовых данных
- Чтение пользовательских настроек
- Доступ к общим конфигурациям

Примеры запросов:

```
SELECT dgrid_get_text('test_key');
```

dgrid_get_bytea

Функция позволяет извлекать ранее сохраненное бинарное значение по ключу из распределенного хранилища.

Основные возможности:

- Извлекает бинарные данные по указанному ключу
- Поддерживает различные пространства хранения (scopes)
- Возвращает NULL, если значение не найдено
- Сохраняет оригинальный формат бинарных данных

Входные параметры:

- key (TEXT) - ключ для поиска значения
- scope (dgrid_scope) - необязательное пространство хранения. По умолчанию: 'USERSCOPE'

Выходные параметры:

- BYTEA - бинарные данные или NULL

Типичные сценарии использования:

- Получение сохраненных бинарных объектов (изображений, документов)
- Восстановление сериализованных данных
- Доступ к общим бинарным ресурсам

- Работа с кешированными бинарными данными

Примеры запросов:

```
SELECT dgrid_get_bytea('test_key');
```

dgrid_has_value

Функция проверяет, существует ли значение по заданному ключу в указанном пространстве (scope). Возвращает TRUE, если значение существует, и FALSE в противном случае.

Основные возможности:

- Проверяет наличие значения по ключу в dgrid
- Работает со всеми типами значений (текстовыми и бинарными)
- Поддерживает различные пространства хранения (scopes)
- Возвращает булево значение (TRUE/FALSE)

Входные параметры:

key (TEXT) - ключ для проверки наличия значения scope (dgrid_scope) - пространство хранения. По умолчанию: USERSCOPE

Выходные параметры:

BOOL: * TRUE - если значение существует * FALSE - если значение отсутствует

Типичные сценарии использования:

- Проверка существования данных перед их получением
- Валидация наличия обязательных настроек
- Проверка кешированных значений
- Мониторинг состояния распределенного хранилища

Примеры запросов:

```
SELECT dgrid_has_value('test_key');
```

dgrid_del_value

Функция удаляет значение по указанному ключу из заданного пространства хранилища.

Основные возможности:

- Удаляет значения по ключу из распределенного хранилища
- Поддерживает различные пространства хранения (scopes)
- Возвращает статус выполнения операции
- Не вызывает ошибку при отсутствии ключа

Входные параметры:

- key (TEXT) - ключ, который требуется удалить
- scope (dgrid_scope) - необязательное пространство хранения. По умолчанию: 'USERSCOPE'

Выходные параметры:

BOOLEAN: * TRUE: если ключ или был успешно удален * FALSE: если ключ не существует в указанной области

Типичные сценарии использования:

- Удаление устаревших данных
- Очистка временных значений
- Управление памятью распределенного хранилища
- Синхронизация данных между системами

Примеры запросов:

```
SELECT dgrid_del_value('test_key');
```