10 Creating Functions

☑Differentiate between a procedure and a function

Function difference: 8.5.1 - 8.5.2

A function has the same structure as a procedure, except that:

- A function heading must include a `RETURN` clause, which specifies the data type of the value that the function returns. (A procedure heading cannot have a `RETURN` clause.)
- In the executable part of a function, every execution path must lead to a `RETURN` statement. Otherwise, the PL/SQL compiler issues a compile-time warning. (In a procedure, the `RETURN` statement is optional and not recommended)
- A function declaration can include these options:
  - `DETERMINISTIC`: Helps the optimizer avoid redundant function invocations
  - `PARALLEL_ENABLE`: Enables the function for parallel execution, making it safe for use in concurrent sessions of parallel DML evaluations
  - `PIPELINED`: Makes a table function pipelined, for use as a row source
  - `RESULT_CACHE`: Stores function results in the PL/SQL function result cache

RETURN statement

- The `RETURN` statement immediately ends the execution of the subprogram or anonymous block that contains it
- A subprogram or anonymous block can contain multiple `RETURN` statements
- > Note: The `RETURN` statement differs from the `RETURN` clause in a function heading, which specifies the data type of the return value
- syntax: `RETURN [expression]`
  - expression:
    - Optional when the `RETURN` statement is in a pipelined table function
    - Required when the `RETURN` statement is in any other function
    - Not allowed when the `RETURN` statement is in a procedure or anonymous block
    - The `RETURN` statement assigns the value of expression to the function identifier. Therefore, the data type of expression must be compatible with the data type in the `RETURN` clause of the function
- **RETURN statement in function**
  - In a function, every execution path must lead to a `RETURN` statement and every `RETURN` statement must specify an expression
  - The `RETURN` statement assigns the value of the expression to the function identifier and returns control to the invoker, where execution resumes immediately after the invocation
  - > Note: In a pipelined table function, a `RETURN` statement need not specify an expression

    - Every execution path in a function must lead to a `RETURN` statement (otherwise, the PL/SQL compiler issues compile-time warning `PLW-05005`)
    - If a function has an execution path without `RETURN` (e.g. if statement with no else clause with return), then the function compiles with warning `PLW-05005: subprogram ... returns without value at line ...`

- **RETURN statement in procedure**
    - In a procedure, the `RETURN` statement returns control to the invoker, where execution resumes immediately after the invocation
    - The `RETURN` statement cannot specify an expression
- **RETURN statement in anonymous block**
    - In an anonymous block, the `RETURN` statement exits its own block and all enclosing blocks
    - The `RETURN` statement cannot specify an expression
    - Example:

```
BEGIN
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Inside inner block.');
    RETURN;
    DBMS_OUTPUT.PUT_LINE('Unreachable statement.');
  END;
  DBMS_OUTPUT.PUT_LINE('Inside outer block. Unreachable statement.');
END;
/
-- result:
Inside inner block.
```

✅Describe the uses of functions

Function result cache: 8.12

Functions that SQL can invoke: 8.13

Function result cache

- When a PL/SQL function has the `RESULT_CACHE` option, its results are cached in the shared global area (SGA) so sessions connected to the same instance can reuse these results when available
- Oracle Database automatically detects all data sources (tables and views) that are queried while a result-cached function is running. If changes to any of these data sources are committed, the cached result becomes invalid across all instances
- The best candidates for result-caching are functions that are invoked frequently but depend on information that changes infrequently or never
- Result-caching avoids redundant computations in recursive functions

Enabling result-caching for a function

- To make a function result-cached, include the `RESULT_CACHE` clause in the function declaration and definition
- The `RESULT_CACHE` clause can appear in the following SQL statements: `CREATE FUNCTION, CREATE TYPE BODY`
- If you declare the function before defining it, you must also include the `RESULT_CACHE` option in the function declaration
- syntax: `RESULT_CACHE [ RELIES_ON ( [data_source [,...]] ) ]`
    - `RELIES_ON`:
        - Specifies the data sources on which the results of the function depend. Each data_source is the name of either a database table or view
        - This clause is deprecated. As of Oracle Database 12c, the database detects all data sources that are queried while a result-cached function is running, and `RELIES_ON` clause does nothing
- restrictions:
    - `RESULT_CACHE` is disallowed on functions with `OUT` or `IN OUT` parameters
    - `RESULT_CACHE` is disallowed on functions with `IN` or `RETURN` parameter of (or containing) these types: `BLOB, CLOB, NCLOB, REF CURSOR`, Collection, Object, Record
    - `RESULT_CACHE` is disallowed on function in an anonymous block
    - `RESULT_CACHE` is disallowed on pipelined table function and nested function

Developing applications with result_cached functions

- When developing an application that uses a result-cached function, make no assumptions about the number of times the body of the function will run for a given set of parameter values
- Some situations in which the body of a result-cached function runs are:
    - The first time a session on this database instance invokes the function with these parameter values
    - When the cached result for these parameter values is invalid When a change to any data source on which the function depends is committed, the cached result becomes invalid
    - When the cached results for these parameter values have aged out If the system needs memory, it might discard the oldest cached values
    - When the function bypasses the cache

Requirements for result-cached functions

- A result-cached PL/SQL function is safe if it always produces the same output for any input that it would produce were it not marked with `RESULT_CACHE`
- This safety is only guaranteed if these conditions are met:
    - When the function is executed, it has no side effects
    - All tables that the function accesses are ordinary, non-SYS-owned permanent tables in the same database as the function
    - The function's result must be determined only by the vector of input actuals together with the committed content, at the current SCN, of the tables that it references
- It is recommended that a result-cached function also meet these criteria:
    - It does not depend on session-specific settings
    - It does not depend on session-specific application contexts

Rules for a cache hit

- Each time a result-cached function is invoked with different parameter values, those parameters and their result are stored in the cache
- Subsequently, when the same function is invoked with the same parameter values (that is, when there is a **cache hit**), the result is retrieved from the cache, instead of being recomputed

Result cache bypass

- In some situations, the cache is bypassed. When the cache is bypassed:
  - The function computes the result instead of retrieving it from the cache
  - The result that the function computes is not added to the cache
- Some examples of situations in which the cache is bypassed are:
  - The cache is unavailable to all sessions
  - A session is performing a DML statement on a table or view on which a result-cached function depends Cache bypass ensures that:
    - The user of each session sees his or her own uncommitted changes
    - The PL/SQL function result cache has only committed changes that are visible to all sessions, so that uncommitted changes in one session are not visible to other sessions

Result cache management

- The PL/SQL function result cache shares its administrative and manageability infrastructure with the Result Cache
- The database administrator manages the server result cache by specifying the `RESULT_CACHE_MAX_SIZE`, `RESULT_CACHE_MAX_RESULT` and `RESULT_CACHE_REMOTE_EXPIRATION` initialization parameters
- The `DBMS_RESULT_CACHE` package provides an interface to allow the DBA to administer that part of the shared pool that is used by the SQL result cache and the PL/SQL function result cache
- Dynamic performance views provide information to monitor the server and client result caches (`V$RESULT_CACHE_STATISTICS, V$RESULT_CACHE_MEMORY, V$RESULT_CACHE_OBJECTS, V$RESULT_CACHE_DEPENDENCY`)

Functions that SQL can invoke

- To be invocable from SQL statements, a stored function (and any subprograms that it invokes) must obey the following purity rules, which are meant to control side effects:
  - When invoked from a `SELECT` statement or a parallelized `INSERT, UPDATE, DELETE, or MERGE` statement, the subprogram cannot modify any database tables
    - > Note: if a select into statement inside a function would return a "no data found" exception, invoking this function inside SQL `SELECT` would not raise any errors but simply return `NULL` data. Invoking this function inside of a PL/SQL block would raise the "no data found" error Discussion about this phenomenon
  - When invoked from an `INSERT, UPDATE, DELETE, or MERGE` statement, the subprogram cannot query or modify any database tables modified by that statement
    - If a function either queries or modifies a table, and a DML statement on that table invokes the function, then `ORA-04091` (mutating-table error) occurs

        - There is one exception: `ORA-04091` does not occur if a single-row `INSERT` statement that is not in a `FORALL` statement invokes the function in a `VALUES` clause
    - When invoked from a `SELECT, INSERT, UPDATE, DELETE, or MERGE` statement, the subprogram cannot execute any of the following SQL statements (unless `PRAGMA AUTONOMOUS_TRANSACTION` was specified):
        - Transaction control statements (such as `COMMIT`)
        - Session control statements (such as `SET ROLE`)
        - System control statements (such as `ALTER SYSTEM`)
        - Database definition language (DDL) statements (such as `CREATE`), which are committed automatically
- If any SQL statement in the execution part of the function violates a rule, then a runtime error occurs when that statement is parsed
- The fewer side effects a function has, the better it can be optimized in a `SELECT` statement, especially if the function is declared with the option `DETERMINISTIC` or `PARALLEL_ENABLE`

☑Work with functions (create, invoke and remove functions)

Function declaration and definition: 13.36

CREATE FUNCTION

ALTER FUNCTION

DROP FUNCTION

Function declaration and definition

- A function is a subprogram that returns a value. The data type of the value is the data type of the function
- A function invocation (or call) is an expression, whose data type is that of the function
- A function declaration is also called a **function specification** or **function spec**
- > Note: this topic applies to nested functions
- syntax:

```
FUNCTION function_name [(parameter_declaration [,...])] RETURN datatype --
declaration
[DETERMINISTIC | PIPELINED | PARALLEL_ENABLE | RESULT_CACHE
[relies_on_clause]]
{IS | AS} {[declare_section] body | call_spec}
```

- function declaration: Declares a function, but does not define it. The definition must appear later in the same block, subprogram, or package as the declaration
    - function heading: The function heading specifies the function name and its parameter list

    - `RETURN` datatype: You cannot constrain this data type (with `NOT NULL`, for example). If datatype is a constrained subtype, then the returned value does not inherit the constraints of the subtype

- function definition: Either defines a function that was declared earlier or both declares and defines a function

CREATE FUNCTION statement

- The `CREATE FUNCTION` statement creates or replaces a standalone function or a call specification
- A standalone function is a function (a subprogram that returns a single value) that is stored in the database
- > Note: A standalone function that you create with the `CREATE FUNCTION` statement differs from a function that you declare and define in a PL/SQL block or package
- A call specification declares a Java method or a C function so that it can be invoked from PL/SQL. You can also use the SQL `CALL` statement to invoke such a method or subprogram
- Prerequisites:
    - To create or replace a standalone function in your schema, you must have the `CREATE PROCEDURE` system privilege
    - To create or replace a standalone function in another user's schema, you must have the `CREATE ANY PROCEDURE` system privilege
    - To invoke a call specification, you may need additional privileges, for example, `EXECUTE` privileges on a C library for a C call specification
    - To embed a `CREATE FUNCTION` statement inside an Oracle precompiler program, you must terminate the statement with the keyword `END-EXEC` followed by the embedded SQL statement terminator for the specific language
- syntax:

```
CREATE [OR REPLACE] [EDITIONABLE | NONEDITIONABLE] FUNCTION
plsql_function_source function_name [(parameter_declaration)]
RETURN datatype [sharing_clause] [subprogram_properties]
{IS | AS} {[declare_section] body | call_spec} ;
```

    - `OR REPLACE`
        - Re-creates the function if it exists, and recompiles it
        - Users who were granted privileges on the function before it was redefined can still access the function without being regranted the privileges
        - If any function-based indexes depend on the function, then the database marks the indexes `DISABLED`
    - `RETURN` datatype
        - For datatype, specify the data type of the return value of the function. The return value can have any data type supported by PL/SQL

        - > Note: Oracle SQL does not support invoking functions with `BOOLEAN` parameters or returns. Therefore, for SQL statements to invoke your user-defined functions, you must design them to return numbers (0 or 1) or character strings ('TRUE' or 'FALSE')
        - The data type cannot specify a length, precision, or scale. The database derives the length, precision, or scale of the return value from the environment from which the function is called
        - If the return type is `ANYDATASET` and you intend to use the function in the `FROM` clause of a query, then you must also specify the `PIPELINED` clause and define a describe method (`ODCITableDescribe`) as part of the implementation type of the function
        - You cannot constrain this data type (with `NOT NULL`, for example)
    - subprogram properties:
        - { invoker_rights_clause | accessible_by_clause | default_collation_clause
          | deterministic_clause | parallel_enable_clause | result_cache_clause | aggregate_clause |
          pipelined_clause | sql_macro_clause }

ALTER FUNCTION statement

- The `ALTER FUNCTION` statement explicitly recompiles a standalone function
- Explicit recompilation eliminates the need for implicit runtime recompilation and prevents associated runtime compilation errors and performance overhead
- Prerequisites:
    - If the function is in the `SYS` schema, you must be connected as `SYSDBA`. Otherwise, the function must be in your schema or you must have `ALTER ANY PROCEDURE` system privilege
- syntax:

```
ALTER FUNCTION function_name
{function_compile_clause | [EDITIONABLE | NONEDITIONABLE]}
```

    - function compile clause

```
COMPILE [DEBUG] [compiler_parameters_clause] [REUSE SETTINGS]
```

DROP FUNCTION statement

- The `DROP FUNCTION` statement drops a standalone function from the database
- Prerequisites
    - The function must be in your schema or you must have the `DROP ANY PROCEDURE` system privilege
- syntax:

```
DROP FUNCTION [schema.] function_name ;
```

- The database invalidates any local objects that depend on, or invoke, the dropped function
- If you subsequently reference one of these objects, then the database tries to recompile the object and returns an error if you have not re-created the dropped function
- If any statistics types are associated with the function, then the database disassociates the statistics types with the `FORCE` option and drops any user-defined statistics collected with the statistics type