Lorem Ipsum Dolor

Table Management Advanced

telung@mac.com

Ordinary Table

- Ordinary Table = Heap-organized Table
- General purpose table
- Data is stored as an unordered collection (heap)

The Row

- Row data is stored in database blocks as variable length records
- Each row has row header, contains the number of columns in the row, the chaining information and the row lock status
- Adjacent rows do not need any space between them

Null Values

- A null is the absence of a value in a column of row.
- A null should not be used to imply any other value, such as zero.
- A column allows null unless a NOT NULL or a PRIMARY KEY integrity constraint has been defined

ROWID

- The ROWID is a pseudo-column that can be queried along with other columns in a table
  - Unique identifier for each row in the database
  - Not stored as a column value
  - Not give the actual physical address of the row
  - The fastest means of accessing a row in a table
  - ROWID needs 10 bytes of storage and is displayed using 18 characters

ROWID

- bbbbbbbb.ssss.ffff
  - bbbbbbbb: Block ID
  - ssss: Block sequence number
  - ffff: file id
  - *DELETE FROM hr.employees E*
    WHERE E.rowid > (SELECT MIN(X.rowid)
    FROM hr.employees X
    WHERE X.employee_id = E.employee_id);

SQL*Plus Commands

- CLEAR SCREEN
- HELP <command>
- SAVE filename[.ext] REPLACE | APPEND
- EXIT

Query Database

- SELECT column_name(s) FROM table_name
  - Can be used to select all the columns
- *SELECT last_name, first_name FROM hr.employees;*

Query Database

- SELECT column FROM table WHERE column operator value

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Using the WHERE Clause

- *SELECT * FROM hr.employees WHERE last_name = 'Bell';*
- *SELECT last_name FROM hr.employees WHERE last_name LIKE 'B%';*
- *SELECT first_name, last_name FROM hr.employees WHERE (first_name = 'Douglas' OR first_name = 'Kimberely') AND last_name = 'Grant';*
- *SELECT DISTINCT last_name FROM hr.employees ORDER BY last_name;*

INSERT INTO

- *SELECT * FROM hr.regions*
- *INSERT INTO hr.regions VALUSE(5, 'Taiwan');*
- *INSERT INTO hr.regions (region_id, region_name) VALUES(6, 'Hong Kong');*

UPDATE

- *UPDATE hr.regions SET region_name = 'Japan' WHERE region_id = 5;*
- *UPDATE hr.locations SET street_address = 'Stien 12', city = 'Stanger' WHERE country_id = 'IN';*

ALTER TABLE

- Use the ALTER TABLE statement to
  - Add a new column
  - Modify an existing column
  - Define a default value for the new column

ALTER TABLE

- *ALTER TABLE hr.employees ADD id NUMBER(6)*
- *ALTER TABLE hr.employees ADD (job VARCHAR2(200));*
- The new column becomes the last column
- *ALTER TABLE hr.employees RENAME COLUMN id TO id_std*
- *ALTER TABLE hr.employees DROP COLUMN id_std*

Modifying a Column

- You can change a column's datatype, size, and default value.
- *ALTER TABLE hr.employees MODIFY (job VARCHAR2(15));*
- A change to the default value affects only subsequent insertions to the table.
- A column must be as wide as the current data
- If a number column already contains data, cannot change precision or scale

Dropping a Column

- Use the DROP COLUMN clause drop columns you no longer need from the table.
- *ALTER TABLE hr.employees DROP COLUMN job;*
- Cannot ROLLBACK

Marking Columns Unused

- Concerned about the length of time if could take to drop column data from all of the rows in a large table
- Use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as UNUSED.

SET UNUSED

- *ALTER TABLE hr.employees ADD job1 varchar2(100), job2 varchar2(50));*
  ALTER TABLE hr.employees SET UNUSED(job1, job2);
  ALTER Table hr.employees SET UNUSED COLUMN job;
- *SELECT * FROM DBA_UNUSED_COL_TABS;*
  ALTER TABLE hr.employees DROP UNUSED COLUMNS;

Delete Data of a Table

- DELETE (DML)
  - Storage space is retained, can ROLLBACK
  - DELETE FROM hr.admin_test
- TRUNCATE TABLE
  - Avoid rollback and increase speed, cannot rollback
  - DDL command
  - Storage space de-allocated
- DROP TABLE

Exercise

- *CREATE TABLE hr.admin_test (job1 varchar2(20), job2 varchar2(40));*
- *begin*
  for i in 1..100 loop
  insert into hr.admin_test values(to_char(i), 'A long test!!!');
  end loop;
  end;
- Another loop, please add to 100,000
- Delete data of job1 >= 100000

Dropping a Table

- All data and structure in the table is deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- Cannot roll back
- *DROP TABLE hr.admin_test CASCADE CONSTRAINTS;*

Changing the Name of an Object

- You must be the owner of the object.
- *RENAME hr.admin_test TO hr.admin_test1;*

Aggregate Functions

- SELECT function(column) FROM table
- AVG
- MAX
- MIN
- COUNT
- SUM
- *SELECT MAX(salary) FROM hr.employees;*
- *SELECT COUNT(*) FROM hr.employees;*
- *SELECT last_name, SUM(salary) FROM hr.employees GROUP BY last_name HAVING SUM(salary) > 15000;*

A Query Based on Multiple Restrictions

- *SELECT first_name, last_name, salary, commission_pct FROM hr.employees*
  WHERE salary > '2000'
  AND commission_pct >= 0.2
  ORDER BY first_name, last_name;

More Complex Functions

- SUM
  *SELECT SUM(salary*commission_pct) FROM hr.employees;*
- AVG
  SELECT first_name, last_name, salary, commission_pct FROM hr.employees WHERE salary > (SELECT AVG(salary) FROM hr.employees) ORDER BY salary DESC;

- GROUP BY
  SELECT department_id, MIN(salary) FROM hr.employees GROUP BY department_id;
- How about show department_name (hr.departments)?

PL/SQL

Building and Managing PL/SQL Program Units

- What Is PL/SQL?
- To answer this question, it is important to remember that every Website you visit, every application you run is constructed from a stack of software technologies. At the top of the stack is the presentation layer, the screens or interactive devices with which the user directly interacts. (These days the most popular languages for implementing presentation layers are Java and .NET.) At the very bottom of the stack is the machine code that communicates with the hardware.
- Somewhere in the middle of the technology stack you will find the database, software that enables us to store and manipulate large volumes of complex data. Relational database technology, built around SQL, is the dominant database technology in the world today.
- SQL is a very powerful, set-oriented language whose sole purpose is to manipulate the contents of relational databases.

https://www.oracle.com/tw/database/technologies/appdev/plsql.html

Procedural Language /Structure Query Language

- Tight integration with SQL
- Improved performance through reduced network traffic
- The front-end code of many applications executes both SQL statements and PL/SQL blocks, to maximize performance while improving the maintainability of those applications.

Building Blocks of PL/SQL

- PL/SQL is a block-structured language.
- A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END, which break up the block into three sections:
  - Declarative: statements that declare variables, constants, and other code elements, which can then be used within that block
  - Executable: statements that are run when the block is executed
  - Exception handling: a specially structured section you can use to "catch," or trap, any exceptions that are raised when the executable section runs
- Only the executable section is required

Examples

- The classic “Hello World!” block contains an executable section that calls the DBMS_OUTPUT.PUT_LINE procedure to display text on the screen:
  BEGIN
  DBMS_OUTPUT.put_line ('Hello World!');
  END;
- Declares a variable of type VARCHAR2 (string) with a maximum length of 100 bytes to hold the string ‘Hello World!’. DBMS_OUTPUT.PUT_LINE then accepts the variable, rather than the literal string, for display:
- ***DECLARE***
  l_message
  VARCHAR2 (100) := 'Hello World!';
  BEGIN
  DBMS_OUTPUT.put_line (l_message);
  END;
- Note that I named the variable l_message. I generally use the l_ prefix for local variables—variables defined within a block of code—and the g_ prefix for global variables defined in a package.

- Nest blocks within blocks as well as the use of the concatenation operator (||) to join together multiple strings.

```
DECLARE
  l_message
  VARCHAR2 (100) := 'Hello';
BEGIN
  DECLARE
     l_message2     VARCHAR2 (100) :=
     l_message || ' World!';
  BEGIN
    DBMS_OUTPUT.put_line (l_message2);
  END;
EXCEPTION
  WHEN OTHERS
  THEN
    DBMS_OUTPUT.put_line
   (DBMS_UTILITY.format_error_stack);
END;
```

Running PL/SQL Blocks

- There are many different tools for executing PL/SQL code. The most basic is SQL*Plus Figure 1 shows an example of executing the simplest of my “Hello World!” example blocks in SQL*Plus.
- ***SET SERVEROUTPUT ON***
- */*

IDE for PL/SQL

- The most popular of these IDEs are:
  - Oracle SQL Developer, from Oracle
  - Toad and SQL Navigator, from Quest Software
  - PL/SQL Developer, from Allround Automations
- Each tool offers slightly different windows and steps for creating, saving, and running PL/SQL blocks as well as enabling and disabling server output.

Name Those Blocks!

- If using anonymous blocks were the only way you could organize your statements, it would be very hard to use PL/SQL to build a large, complex application.
- PL/SQL supports the definition of named blocks of code, also known as subprograms. Subprograms can be procedures or functions. Generally, a procedure is used to perform an action and a function is used to calculate and return a value.

create a procedure named hello_world by executing the following data definition language (DDL) command:

```
CREATE OR REPLACE PROCEDURE
hello_world
IS
  l_message
  VARCHAR2 (100) := 'Hello World!';
BEGIN
  DBMS_OUTPUT.put_line (l_message);
END hello_world;
```

- I have now, in effect, extended PL/SQL. In addition to calling programs created by Oracle and installed in the database (such as DBMS_OUTPUT.PUT_LINE), I can call my own subprogram inside a PL/SQL block:

```
BEGIN
   hello_world;
END;
```

Procedure 的好處

- I have hidden all the details of how I say hello to the world inside the body, or implementation, of my procedure.
- I can now call this hello_world procedure and have it display the desired message without having to write the call to DBMS_OUTPUT .PUT_LINE or figure out the correct way to format the string.
- I can call this procedure from any location in my application. So if I ever need to change that string, I will do so in one place, the single point of definition of that string.
- Parameters pass information into subprograms when they are called, and they enable you to create subprograms that are more flexible and generic. They can be used in many different contexts.

Procedure with parameter

- Pass the changing parts as parameters and have a single procedure that can be used under different circumstances:

```
CREATE OR REPLACE PROCEDURE
hello_place (place_in IN VARCHAR2)
IS
  l_message  VARCHAR2 (100);
BEGIN
  l_message  := 'Hello ' || place_in;
  DBMS_OUTPUT.put_line (l_message);
END hello_place;
```

- Right after the name of the procedure, I add open and close parentheses, and inside them I provide a single parameter. I can have multiple parameters, but each parameter follows the same basic form:
  parameter_name parameter_mode datatype
- You must, in other words, provide a name for the parameter, the mode or way it will be used (IN = read only), and the type of data that will be passed to the subprogram through this parameter.

Call procedure with parameter

- In this case, I am going to pass a string for read-only use to the hello_place procedure.

- And I can now say hello to my world and my universe as follows:

```
BEGIN
   hello_place ('World');
   hello_place ('Universe');
END;
```

Function - subprogram

- Now suppose I don't just want to display my "Hello" messages. Sometimes I need to save those messages in a database table; at other times, I must pass the string back to the host environment for display in a Web browser.
- I can achieve this desired level of flexibility by moving the code that constructs the message into its own function:

```
CREATE OR REPLACE FUNCTION
hello_message
   (place_in IN VARCHAR2)
    RETURN VARCHAR2
IS
BEGIN
   RETURN 'Hello ' || place_in;
END hello_message;
```

Call function

- This subprogram differs from the original procedure as follows:
  - The type of program is now FUNCTION, not PROCEDURE.
  - The subprogram name now describes the data being returned, not the action being taken.
  - The body or implementation of the subprogram now contains a RETURN clause that constructs the message and passes it back to the calling block.
  - The RETURN clause after the parameter list sets the type of data returned by the function.
- With the code needed to construct the message inside the hello_message function, I can use this message in multiple ways. I can, for example, call the function to retrieve the message and assign it to a variable:

```
DECLARE
  l_message  VARCHAR2 (100);
BEGIN
  l_message := hello_message ('Universe');
END;
```