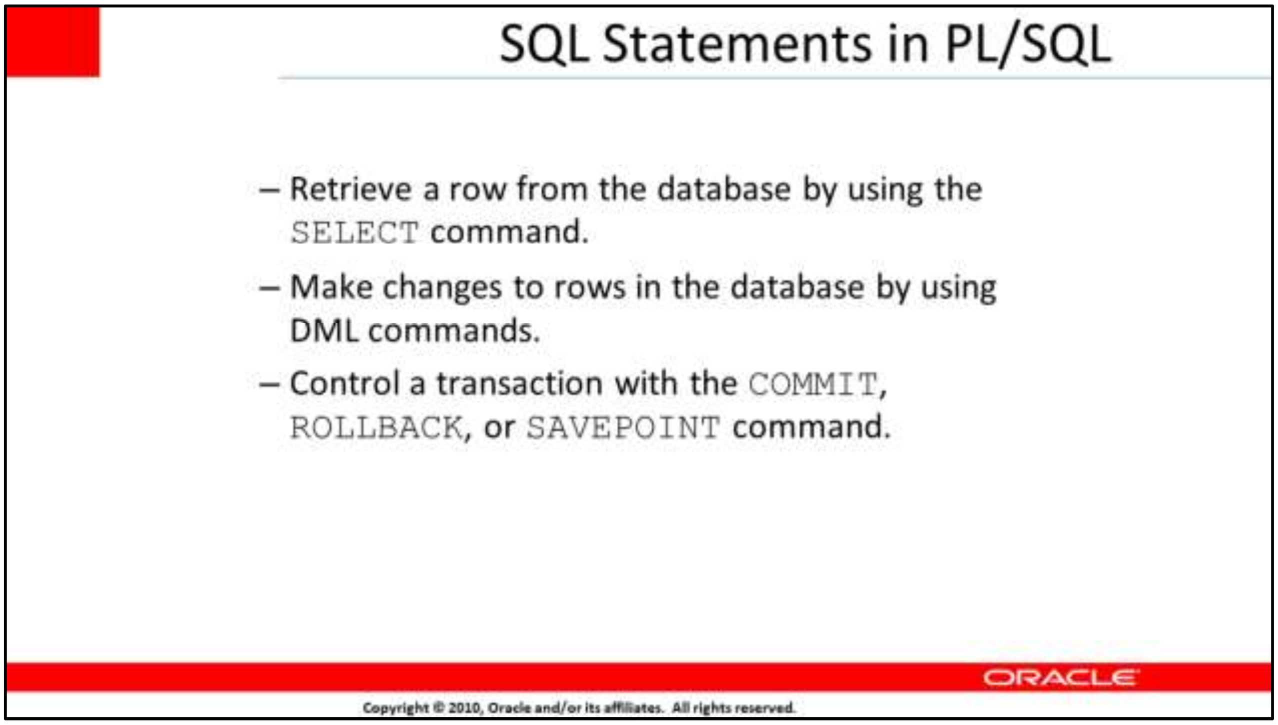

SQL Statements in PL/SQL

- Retrieve a row from the database by using the `SELECT` command.
- Make changes to rows in the database by using DML commands.
- Control a transaction with the `COMMIT`, `ROLLBACK`, or `SAVEPOINT` command.

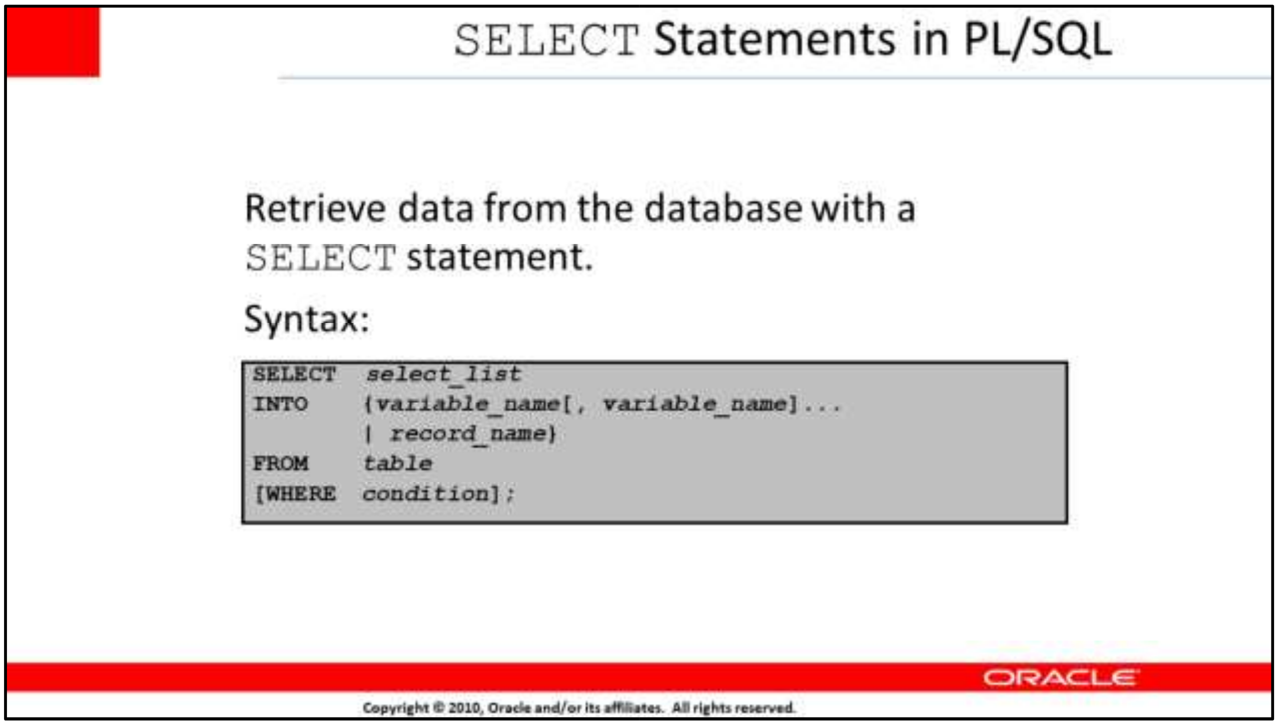
SQL Statements in PL/SQL

In a PL/SQL block, you use SQL statements to retrieve and modify data from the database table. PL/SQL supports data manipulation language (DML) and transaction control commands. You can use DML commands to modify the data in a database table. However, remember the following points while using DML statements and transaction control commands in PL/SQL blocks: The END keyword signals the end of a PL/SQL block, not the end of a transaction. Just as a block can span multiple transactions, a transaction can span multiple blocks.

PL/SQL does not directly support data definition language (DDL) statements such as CREATETABLE, ALTERTABLE, or DROP TABLE. PL/SQL supports early binding, which cannot happen if applications have to create database objects at run time by passing values. DDL statements cannot be directly executed. These statements are dynamic SQL statements. Dynamic SQL statements are built as character strings at run time and can contain placeholders for parameters. Therefore, you can use dynamic SQL to execute your DDL statements in PL/SQL. The details of working with dynamic SQL are covered in the course titled Oracle Database: Develop PL/SQL Program

Units.

PL/SQL does not directly support data control language (DCL) statements such as GRANT or REVOKE. You can use dynamic SQL to execute them.



SELECT Statements in PL/SQL

Retrieve data from the database with a SELECT statement.

Syntax:

```
SELECT   select_list
INTO     {variable_name[, variable_name]...
         | record_name}
FROM     table
[WHERE   condition];
```

SELECT Statements in PL/SQL

Use the SELECT statement to retrieve data from the database.

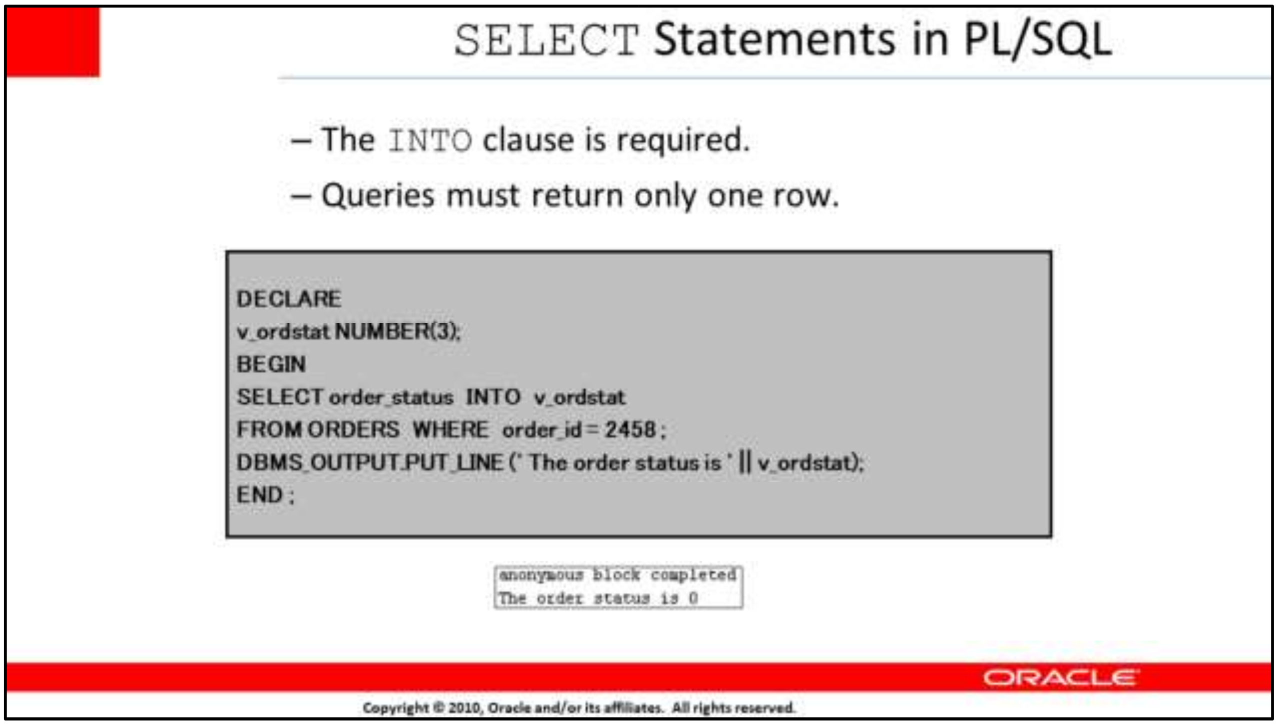
Specify the same number of variables in the INTO clause as the number of database columns in the SELECT clause. Be sure that they correspond positionally and that their data types are compatible. Use group functions, such as SUM, in a SQL statement, because group functions apply to groups of rows in a table.

select_list	List of at least one column; can include SQL expressions, row functions, or group functions
variable_name	Scalar variable that holds the retrieved value
record_name	PL/SQL record that holds the retrieved values
table	Specifies the database table name

conditionGuidelines for Retrieving Data in PL/SQLIs composed of column names, expressions, constants, and

Terminate each SQL statement with a semicolon (comparison operators, including PL/SQL variables and constants;). Every value retrieved must be stored in a variable by using the INTO clause.

The WHERE clause is optional and can be used to specify input variables, constants, literals, and PL/SQL expressions. However, when you use the INTO clause, you should fetch only one row; using the WHERE clause is required in such cases.



SELECT Statements in PL/SQL

- The INTO clause is required.
- Queries must return only one row.

```
DECLARE
v_ordstat NUMBER(3);
BEGIN
SELECT order_status INTO v_ordstat
FROM ORDERS WHERE order_id = 2458 ;
DBMS_OUTPUT.PUT_LINE (' The order status is ' || v_ordstat);
END ;
```

```
anonymous block completed
The order status is 0
```

SELECT Statements in PL/SQL (continued)

INTO Clause

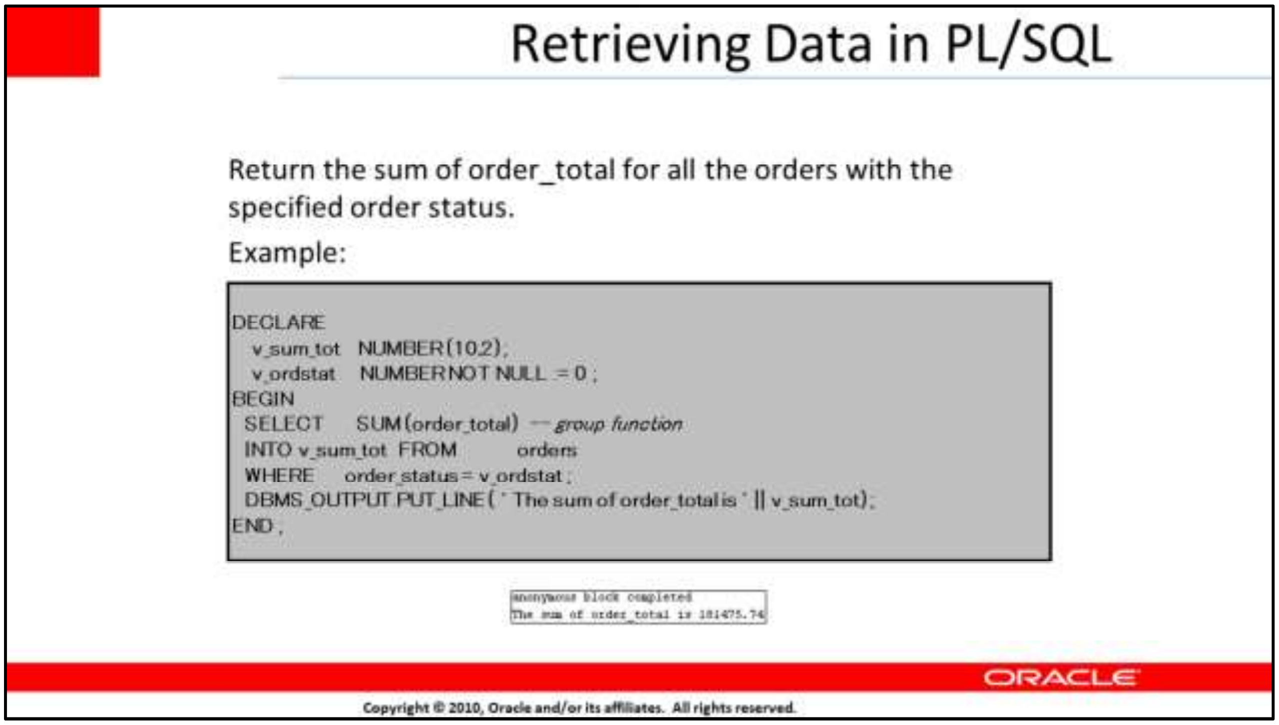
The INTO clause is mandatory and occurs between the SELECT and FROM clauses. It is used to specify the names of variables that hold the values that SQL returns from the SELECT clause. You must specify one variable for each item selected, and the order of the variables must correspond with the items selected. Use the INTO clause to populate either PL/SQL variables or host variables.

Queries Must Return Only One Row

SELECT statements within a PL/SQL block fall into the ANSI classification of embedded SQL, for which the following rule applies: Queries must return only one row. A query that returns more than one row or no row generates an error. PL/SQL manages these errors by raising standard exceptions, which you can handle in the exception section of the block with the NO_DATA_FOUND and TOO_MANY_ROWS exceptions. Include a WHERE condition in the SQL statement so that the statement returns a single row. You learn about exception handling in the lesson titled “Handling Exceptions.”

Note: In all cases where DBMS_OUTPUT.PUT_LINE is used in the code examples, the SET SERVEROUTPUTON statement precedes the block.

How to Retrieve Multiple Rows from a Table and Operate on the Data A SELECT statement with the INTO clause can retrieve only one row at a time. If your requirement is to retrieve multiple rows and operate on the data, you can make use of explicit cursors. You are introduced to cursors later in this lesson and learn about explicit cursors in the lesson titled “Using Explicit Cursors.”



Retrieving Data in PL/SQL

Return the sum of order_total for all the orders with the specified order status.

Example:

```
DECLARE
  v_sum_tot   NUMBER(10,2);
  v_ordstat   NUMBER NOT NULL := 0 ;
BEGIN
 SELECT    SUM(order_total) -- group function
 INTO v_sum_tot  FROM        orders
 WHERE    order_status = v_ordstat ;
 DBMS_OUTPUT.PUT_LINE( ' The sum of order_total is ' || v_sum_tot);
END ;
```

```
anonymous block completed
The sum of order_total is 181475.74
```

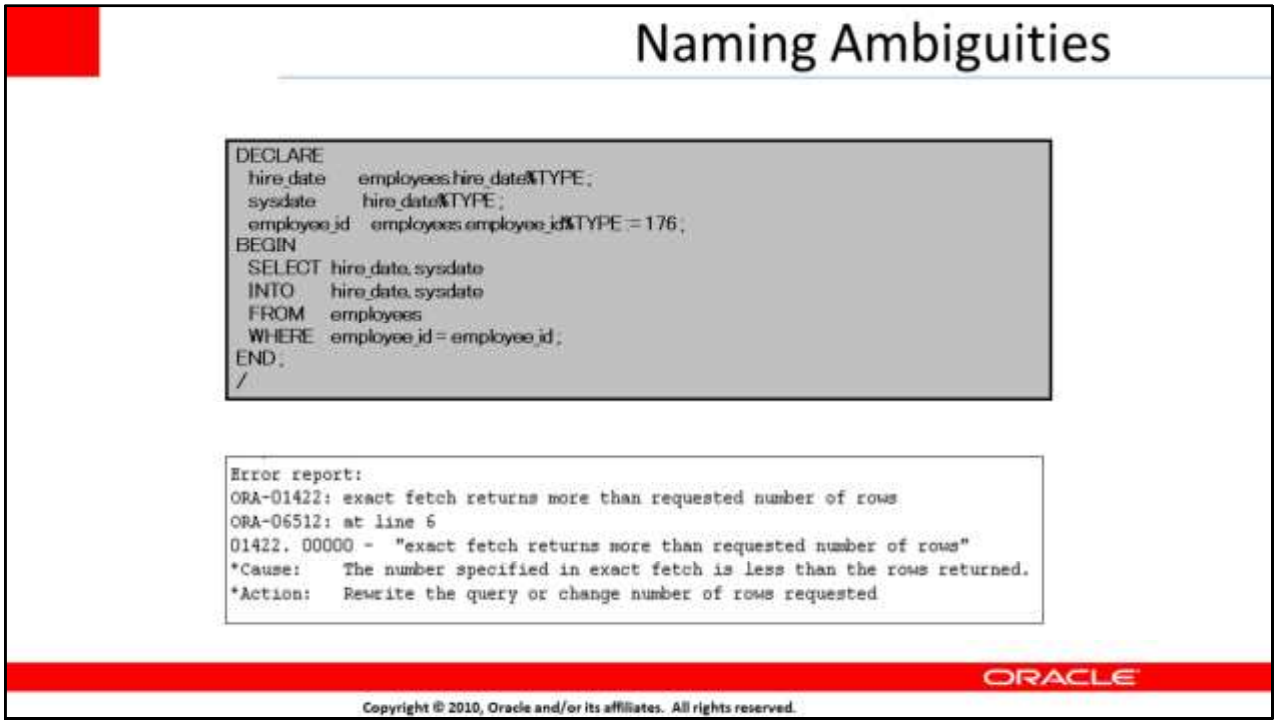
Retrieving Data in PL/SQL

In the example in the slide, the v_ord_date and v_ord_total variables are declared in the declarative section of the PL/SQL block. In the executable section, the values of the order_date and order_total columns for the order with the order_id 2381 are retrieved from the orders table. Next, they are stored in the ord_date and ord_total variables, respectively. Observe how the INTO clause, along with the SELECT statement, retrieves the database column values and stores them in the PL/SQL variables.

Retrieving Data in PL/SQL (continued)

In the example in the slide, the v_sum_tot and v_ordstat variables are declared in the declarative section of the PL/SQL block. In the executable section, the sum of order_total for the orders with order_status 0 is computed using the SQL aggregate function SUM. The calculated total salary is assigned to the v_sum_tot variable.

Note: Group functions cannot be used in PL/SQL syntax. They must be used in SQL statements within a PL/SQL block as shown in the example in the slide.



Naming Ambiguities

```
DECLARE
  hire_date     employees.hire_date%TYPE;
  sysdate        hire_date%TYPE;
  employee_id   employees.employee_id%TYPE := 176;
BEGIN
  SELECT  hire_date, sysdate
  INTO      hire_date, sysdate
  FROM     employees
  WHERE   employee_id = employee_id;
END;
/
```

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
01422. 00000 -  "exact fetch returns more than requested number of rows"
*Cause:    The number specified in exact fetch is less than the rows returned.
*Action:   Rewrite the query or change number of rows requested
```

For instance, you cannot use group functions using the following syntax:

V_sum_sal := SUM(employees.salary);

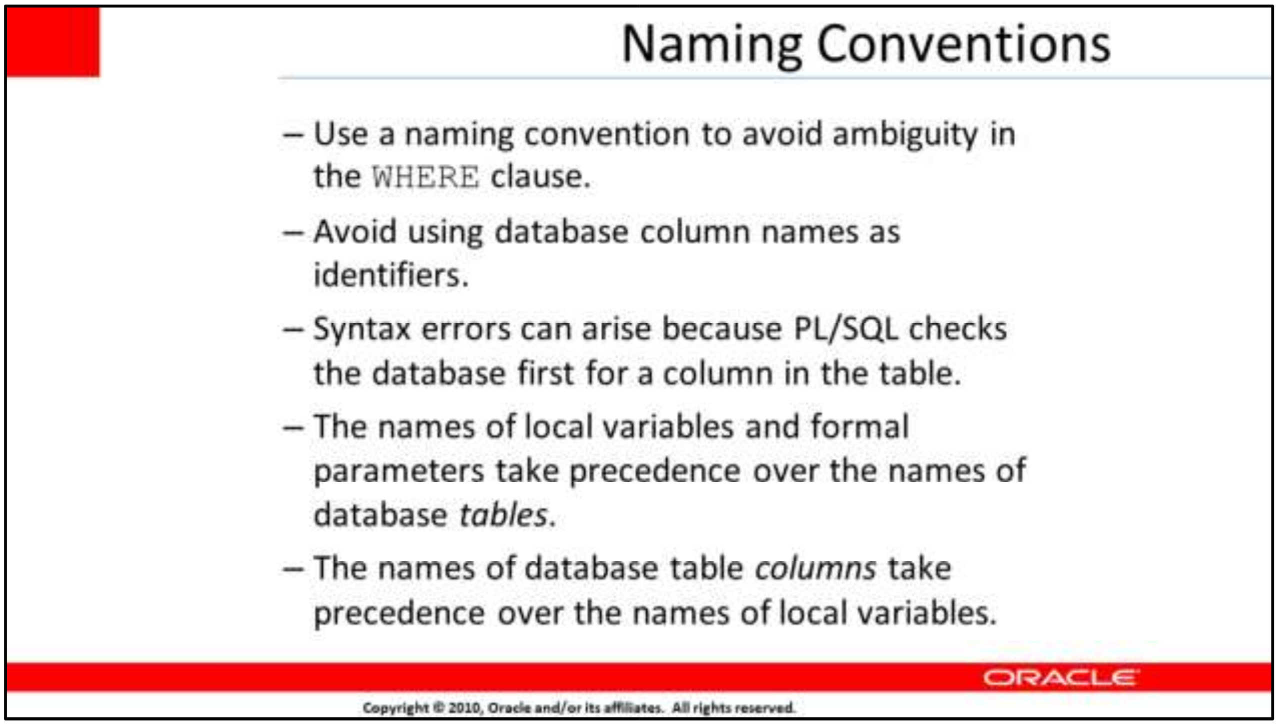
Naming Ambiguities

In potentially ambiguous SQL statements, the names of database columns take precedence over the names of local variables.

The example shown in the slide is defined as follows: Retrieve the hire date and today's date from the employees table for employee_id176. This example raises an unhandled run-time exception because, in the WHERE clause, the PL/SQL variable names are the same as the database column names in the employees table.

The following DELETE statement removes all employees from the employees table, where the last name is not null (not just "King"), because the Oracle Server assumes that both occurrences of last_name in the WHERE clause refer to the database column:

```
DECLARE
 last_name VARCHAR2(25) := 'King';
BEGIN
```



Naming Conventions

- Use a naming convention to avoid ambiguity in the WHERE clause.
- Avoid using database column names as identifiers.
- Syntax errors can arise because PL/SQL checks the database first for a column in the table.
- The names of local variables and formal parameters take precedence over the names of database *tables*.
- The names of database table *columns* take precedence over the names of local variables.

```
  DELETE FROM employees WHERE last_name =
last_name;
. . .
```

Naming Conventions

Avoid ambiguity in the WHERE clause by adhering to a naming convention that distinguishes database column names from PL/SQL variable names.

Database columns and identifiers should have distinct names. Syntax errors can arise because PL/SQL checks the database first for a column in the table.

Note: There is no possibility of ambiguity in the SELECT clause because any identifier in the SELECT clause must be a database column name. There is no possibility of ambiguity in the INTO clause because identifiers in the INTO clause must be PL/SQL variables. The possibility of confusion is present only in the WHERE clause.