Test 1 - Introduction to Oracle: SQL and PL/SQL (Exam #1Z0-001)

Selecting Data
The most common manipulation of data in the Oracle database is to select it, and the means by which to select data from Oracle is the **select** statement. The **select** statement has two basic parts, the **select** clause and the **from** clause.

The **select** clause identifies the column of the table that the user would like to view contents of. The **from** clause identifies the table in which the data selected is stored.

Performing Arithmetic Equations

Often, users will want to perform calculations involving the data selected from a table. Oracle allows for basic, intermediate, and complex manipulation of data selected from a database table through the use of standard arithmetic notation such as plus (+), minus(-), multiply (*), and divide (/). These operators can be used to perform math calculations on the data **select**ed from a table or as math operators on numbers in calculator-like fashion. In order to perform calculations on numbers that are not selected from any table, the user must utilize the DUAL table. DUAL is simply an empty table with one column that fulfills the syntactic requirements of SQL statements like **select**, which need a table name in the **from** clause in order to work.

Handling NULL Values

When manipulating data from a table, the user must remember to handle cases when column data for a particular row is nonexistent. Nonexistent column data in a table row is often referred to as being NULL. These NULL values can be viewed either as blank space, by default, or the user can account for the appearance of null data by using a special function that will substitute null fields with a data value. The name of this special function is **nvl( )**. The **nvl( )** function takes two parameters: the first is the column or value to be investigated for being null, and the second is the default value **nvl( )** will substitute if the column or value is null. The **nvl( )** function operates on all sorts of datatypes, including CHAR, VARCHAR2, NUMBER, and DATE.

"Renaming" Columns with Column Aliases

When performing special operations on columns in a **select** statement, Oracle often displays hard-to-read headings for the column name because Oracle draws the column name directly from the **select** clause of the **select** statement. The user can avoid this problem by giving a column alias for Oracle to use instead. For example, the following **select** may produce a cryptic column heading: **select nvl(empid,'00000') EMPID ...**, while a column alias would allow Oracle to provide a more meaningful heading: **select nvl(empid,'00000') EMPID ...**. Column aliases are specified as character strings following the function and/or column name the alias will substitute. Be sure to include white space between the function and/or column name and the alias.

Putting Columns Together with Concatenation

Columns can be concatenated together using the double-pipe (||) delimiter. This operation is useful for placing information closer together, or to use special characters to separate the output, such as commas or others.

*Editing SQL Queries Within SQL*Plus*

The SQL statement itself is entered using the SQL*Plus tool. If a user makes an error while typing in the line of SQL, the user can use the backspace key to erase characters until he or she reaches the mistake; however, this approach only works if the user is still on the same line in the SQL entry buffer. If the user has already proceeded to another line, or if he or she has already tried to execute the command, then he or she can type in the number corresponding to the line to be corrected to **select** that line for editing. Then, the user can type in the change command, abbreviated **c/old/new**, where **old** is the existing version of the string containing the mistake, and **new** is the correction. If this all sounds complicated, the user can simply type **edit**, or **ed** from the prompt in SQL*Plus, and Oracle will immediately bring up the user's favorite text editor. The text editor used here can be specified or changed with the **define_editor**="*youreditor*" command.

Limiting Selected Output

The number or order of selected rows from the database can be limited with various options.

The Order by clause

This is a clause that allows the user to specify two things-the first is a column on which to list the data in order, the second is whether Oracle should use ascending or descending order. Usage of the **order by** clause can make output from an Oracle **select** statement more readable, since there is no guarantee that the data in Oracle will be stored in any particular order.

The Where close

The second means of limiting selected output is the **where** clause. Proper use of this clause is key to successful usage of Oracle and SQL. In the **where** clause, the user can specify one or more comparison criteria that must be met by the data in a table in order for Oracle to select the row. A comparison consists of two elements that are compared using a comparison operator, which may consist of a logic operator such as equality (=), inequality (<>,!=, or ^=), less than (<) or greater than (>), or a combination of less or greater than and equality. Alternately, the user can also utilize special comparison operators that enable for pattern matches using **like %**, range scans using **between** x **and** y, or fuzzy logic with the **soundex(x) = soundex(y)** statement. In addition, one or more comparison operations may be specified in the **where** clause, joined together with **and** or the **or** operator, or preceded by **not**.

Using Single-row Functions

Data selected in Oracle can be modified with the use of several functions available in Oracle. These functions may work on many different types of data, as is the case with **nvl( )** other functions called **decode( )**, **greatest( )**, or **least( )**. Alternately, their use may be limited to a particular datatype. These functions may be divided into categories based on the types of data they can handle. Typically, the functions are categorized into text or character functions, math or number functions, and date functions.

Various Single-row Functions Explained

Related to the set of functions designed to convert data from one thing to another are several functions that manipulate text strings. These functions are similar in concept to **nvl( )** and **decode( )** in that they can perform a change on a piece of data, but the functions in this family can perform data change on only one type of data-text. As such, the functions in this family are often referred to as text, or character functions. In this family are several functions in Oracle, for which some of the highlighted functions that are most used are listed below.

lpad(x,y[,z]) **rpad(x,y[,z])**	Returns the column "padded" on the left or right side of the data in the column passed as x to a width passed as y. The optional passed value z indicates the character that **lpad** or **rpad** will insert into the column.
lower(x) **upper(x)** **initcap(x)**	Returns the column value passed as x into all lowercase or uppercase, or changes the initial letter in the string to a capital letter.
length(x)	Returns a number indicating the number of characters in the column value passed as x.
substr(x,y[,z])	Returns a substring of string x, starting at character number y to the end, which is optionally defined by the character appearing in position z of the string.

Others are designed to perform specialized mathematical functions such as those used in scientific applications like sine and logarithm, which should already be fairly well understood by those with a background in trigonometry. These operations are commonly referred to as math or number operations. The functions falling into this category are listed below. These functions are not all the ones available in Oracle, but rather are the most commonly used ones that will likely be used on OCP Exam 1.

abs(x)	Obtains the absolute value for a number. For example, the absolute value of (-1) is 1, while the absolute value of 6 is 6.
ceil(x)	Similar to executing **round** (see below) on an integer (i.e., **round(x,0)**, except **ceil** always rounds up. For example, **ceil(1.6)** = 2. Note that rounding "up" on negative numbers produces a value closer to zero (e.g., **ceil(-1.6)** = -1, not -2).
floor(x)	Similar to **ceil** (see above), except **floor** always rounds down. For example, **floor(1.6)** = 1. Note that rounding "down" on negative numbers produces a value further away from zero (e.g., **floor(-1.6)** = -2, not -1.
mod(x,y)	The modulus of x, as defined by long division as the integer remainder left over when x is divided by y until no further whole number can be produced. An example is **mod(10,3) =** 1, or **mod(10,2) =** 0.
round(x,y)	Round x to the decimal precision of y. If y is negative, round to the precision of y places to the left of the decimal point. For example, **round(134.345,1)** = 134.3, **round(134.345,0)** = 134, **round(134.345,-1)** = 130.
sign(x)	Displays **integer** value corresponding to the sign of x, 1 if x is positive, -1 if x is negative.
sqrt(x)	The square root of x.
trunc(x,y)	Truncate value of x to decimal precision y. If y is negative, then truncate to y number of places to the left of the decimal point.
vsize(x)	The storage size in bytes for value x.

The final category of number functions that is the set of list functions. These functions are actually used for many different datatypes, including text, numeric, and date. The list functions are listed below.

greatest(x,y,...)	Returns the highest value from list of text strings, numbers, or dates (x,y…).
least(x,y,...)	Returns the lowest value from list of text strings, numbers, or dates (x,y…).

Another class of data functions available in Oracle correspond to another commonly used datatype in the Oracle database-the DATE datatype. The functions that perform operations on dates are known as date functions. Before diving into the functions, a useful item in Oracle related to dates will be presented. There is a special keyword that can be specified to give Oracle users the current date. This keyword is called **sysdate**. The functions that can be used on DATE columns are listed in the following definitions:

add_months(x,y)	Returns a date corresponding to date x plus y months.
last_day(x)	Returns the date of the last day of the month that contains date x.
months_between(x,y)	Returns a number of months between y and x as produced by y-x. Can return a decimal value.
new_time(x,y,z)	Returns the current date and time for date x in time zone y as it would be in time zone z.

Why use functions at all? The functions available in Oracle are highly useful for executing well-defined operations on data in a table or constant values in an easy way. For example, if the user were working with a scientific organization to produce a report of data for that organization, the user may want to use some of the math functions available in Oracle. Rather than selecting data from a table and performing standard mathematical calculations using a scientific calculator, the user may instead execute the functions on that data and produce the report cleanly, in one step. The use of functions in Oracle often saves time and energy.

Conversion functions

Still other functions are designed to convert columns of one datatype to another type. As these functions are simply designed to change the datatype of the column value, not actually modify the data itself, the functions are called conversion functions. There are several different conversion functions available in the Oracle database. The ones available appear in the following list:

to_char(x)	Converts noncharacter value x to character
to_number(x)	Converts nonnumeric value x to number
to_date(x[,y])	Converts nondate value x to date, using format specified by y
to_multi_byte(x)	Converts single-byte character string x to multibyte characters according to national language standards
to_single_byte(x)	Converts multibyte character string x to single-byte characters according to national language standards
chartorowid(x)	Converts string of characters x into an Oracle ROWID
rowidtochar(x)	Converts string of characters x into an Oracle ROWID
hextoraw(x)	Converts hexadecimal (base-16) value x into raw (binary) format
rawtohex(x)	Converts raw (binary) value x in to hexadecimal (base-16) format

convert(x[,y[,z]])	Executes a conversion of alphanumeric string x from the current character set optionally specified as z to the one specified by y
translate(x,y,z)	Executes a simple value conversion for character or numeric string x into something else based on the conversion factors y and z

Advanced Data Selection in Oracle

Displaying Data from Multiple Tables

Data in a table can be linked if there is a common or shared column between the two tables. This shared column is often referred to as a foreign key. Foreign keys establish a relationship between two tables that is referred to as a parent/child relationship. The parent table is typically the table in which the common column is defined as a primary key, or the column by which uniqueness is identified for rows in the table. The child table is typically the table in which the column is not the primary key, but refers to the primary key in the parent table.

Types of joins

There are two types of joins. One of those types is the "inner" join, also known as an equijoin. An inner join is a data join based on equality comparisons between common columns of two or more tables. An "outer" join is a nonequality join operation that allows the user to obtain output from a table even if there is no corresponding data for that record in the other table.

Select Statements that Join Data from More than One Table

Joins are generated by using select statements in the following way. First, the columns desired in the result set are defined in the select clause of the statement. Those columns may or may not be preceded with a table definition, depending on whether or not the column appears in more than one table. If the common column is named differently in each table, then there is no need to identify the table name along with the column name, as Oracle will be able to distinguish which table the column belongs to automatically. However, if the column name is duplicated in two or more tables, then the user must specify which column he or she would like to obtain data from, since Oracle must be able to resolve any ambiguities clearly at the time the query is parsed. The columns from which data is selected are named in the from clause, and may optionally be followed by a table alias. A table alias is similar in principle to a column alias. The where clause of a join statement specifies how the join is performed.

Creation of inner and outer join

An inner join is created by specifying the two shared columns in each table in an equality comparison. An outer join is created in the same way, with an additional special marker placed by the column specification of the "outer" table, or the table in which there need not be data corresponding to rows in the other table for that data in the other table to be returned. That special marker is indicated by a (+).

Joining a Table to Itself

Finally, a table may be joined to itself with the use of table aliases. This activity is often done to determine if there are records in a table with slightly different information from rows that otherwise are duplicate rows.

Group Functions and Their Uses

Another advanced technique for data selection in Oracle databases is the use of grouping functions. Data can be grouped together in order to provide additional meaning to the data. Columns in a table can also be treated as a group in order to perform certain operations on them. These grouping functions often perform math operations such as averaging values or obtaining standard deviation on the dataset. Other group functions available on groups of data are max( ), min( ), sum( ), and count( ).

Using the GROUP BY Clause

One common grouping operation performed on data for reporting purposes is a special clause in select statements called group by. This clause allows the user to segment output data and perform grouping operations on it. There is another special operation associated with grouping that acts as a where clause for which to limit the output produced by the selection. This limiting operation is designated by the having keyword. The criteria for including or excluding data using the having clause can be identified in one of two ways. Either criterion can be a hard-coded value or it can be based on the results of a select statement embedded into the overarching select statement. This embedded selection is called a subquery.

Using Subqueries

Another advanced function offered by select statements is the use of subqueries in the where clause of the select statement. A select statement can have 16 or more nested subqueries in the where clause, although it is not generally advisable to do so based on performance. Subqueries allow the user to specify unknown search criteria for the comparisons in the where clause as opposed to using strictly hard-coded values. Subqueries also illustrate the principle of data scope in SQL statements by virtue of the fact that the user can specify columns that appear in the parent query, even when those columns do not appear in the table used in the subquery.

Another use of subqueries can be found in association with a special operation that can be used in the where clause of a select statement. The name of this special operation is exists. This operation produces a TRUE or FALSE value based on whether or not the related subquery produces data. The exists clause is a popular option for users to incorporate subqueries into their select statements.

Output from the query can be placed into an order specified by the user with the assistance of the order by clause. However, the user must make sure that the columns in the order by clause are the same as those actually listed by the outermost select statement. The order by clause can also be used in subqueries; however, since the subqueries of a select statement are usually used to determine a valid value for searching or as part of an exists clause, the user should be more concerned about the existence of the data than the order in which data is returned from the subquery. Therefore, there is not much value added to using the order by clause in subqueries.

Using Runtime Variables

One final advanced technique is the specification of variables at run time. This technique is especially valuable in order to provide reusability in a data selection statement. In order to denote a runtime variable in SQL, the user should place a variable name in the comparison operation the user wants to specify a runtime value for. The name of that variable in the select statement should be preceded with a special character to denote it as a variable. By default, this character is an ampersand (&). However, the default variable specification character can be changed with the use of the set define command at the prompt.

Runtime variables can be specified for SQL statements in other ways as well. The define command can be used to identify a runtime variable for a select statement automatically. After

being defined and specified in the define command, a variable is specified for the entire session or until it is altered with the undefine command. In this way, the user can avoid the entire process of having to input values for the runtime variables. The final technique covered on select statements is the usage of accept to redefine the text displayed for the input prompt. More cosmetic than anything else, accept allows the user to display a more direct message than the Oracle default message for data entry.

NEW FEATURES IN SQL*Plus 8.0

CONNECT command incites the user to change an expired password.

EXIT command has a :BindVariable clause and thus, can be referenced in PL/SQL.

The limits of **CLOB** and **NCLOB** datatypes is determined by LONG and **LONGCHUNKSIZE** datatypes.

ATTRIBUTE command. For a given column, the attributes are displayed.

SET command has a **LOBOFFSET** clause that sets the beginning position from which **CLOB** and **NCLOB** data is returned.

A **NONE** clause is attached to the **SET NEWPAGE** command. Its function is to disallow blank lines and formfeed between pages during printing.

PASSWORD command, passwords can be modified with no echo of password to the input device.

VARIABLE command comprises clauses **NCHAR, NVARCHAR2 (NCHAR VARYING), CLOB** and **NCLOB.**

4000 is the maximum length of **VARCHAR2** and **NVARCHAR2**. The maximum length of **CHAR** and **NCHAR** bind variables are 2000.

Overview of Data Modeling and Database Design

Stages of System Development

In order to create a database in Oracle, it is important that all stages of system development be executed carefully. Some of the stages include needs assessment, requirements definition, database design, application development, performance tuning, security enforcement, and enhancements development. The final stage in that life cycle is really a miniature version of the first several stages rolled into one.

The needs assessment stage is a critical one. It is the period of time where the users of the system are identified, and the desired and required features of the system are documented. After needs assessment, a full list of requirements should be agreed upon and documented so as to avoid costly rework later. Once the requirements of the system are completely understood, the developers of the database portion of the application should model the business process required into an entity-relationship diagram, which consists of entities, or persons, places, things, or ideas involved in the process flow, and the relationships between each entity. This entity-relationship diagram will then be used to create a logical data model, or a pictorial diagram of the tables that will represent each entity and the referential integrity constraints that will represent each relationship. Ordinality is a key point here.

Ordinality defines whether the relationship is mandatory for the entities partaking of the relationship, and the record-to-record correspondence of one record in a database. There are three types of record-to-record correspondence in the database-one-to-one, one-to-many, and many-to-many.

A one-to-one correspondence means that one record of one table corresponds to one record in another.

One-to-many correspondence means that one record from one table corresponds to many records of another table.

Many-to-many correspondence means that several records from one table correspond to several records on another table.

Creating the Tables of an Oracle Database

Once the planning is complete, than developers and DBAs can move forward with the process of actually creating the database. A table can be created with several different columns. The allowed datatypes for these columns in Oracle7 are VARCHAR2, CHAR, NUMBER, DATE, RAW, LONG, LONG RAW, MLSLABEL and ROWID. More datatypes are available in Oracle8. One or more of these columns is used to define the primary key, or element in each row that distinguishes one row of data from another in the table.

A primary key is one type of integrity constraint. Another type of integrity constraint is the foreign key, which defines referential integrity on the table, creating table relationships and often modeling the relationships between entities from the entity-relationship diagram. Referential integrity produces a parent/child relationship between two tables.

Using Table Naming Conventions

Sometimes it is useful to name tables according to conventions that have the child objects take on the name of the parent object as part of their own name. The three other constraints available on the database are unique, check, and NOT NULL. Unique constraints prevent duplicate non-NULL values from appearing in a column for two or more rows. Check constraints verify data in a column against a set of constants defined to be valid values. NOT NULL constraints prevent the entry of NULL data for a column on which the NOT NULL constraint is defined. Two of the five constraints create indexes to help enforce the integrity they are designed to enforce. Those two constraints are the ones designed to enforce uniqueness, the unique constraint and the primary key. Finally, a table is created with no data in it, except in the case of the **create table as select**. This statement allows the user to create a table with row data prepopulated from another table. All options available for regular **select** statements are available in this statement as well.

The Oracle Data Dictionary

The data dictionary contains information about all objects created in the database. It also contains a listing of available columns in each object created in the database. Information about table columns can be obtained using the **describe** command, followed by the name of the table the user wants to view the columns on. Information is kept in data dictionary tables about the objects created in Oracle, where they are stored, and performance statistics. However, the user will not usually access the tables of the data dictionary directly. Rather, the user generally will look at that data using data dictionary views. Data can be selected from views in the same way it can be selected from tables. No user is able to **delete** data from the data dictionary, because doing so could permanently damage the Oracle database. All tables and views in the Oracle data dictionary are owned by SYS.

Available Dictionary Views & Querying the Data Dictionary

Those views are divided into three general categories that correspond to the scope of data availability in the view. The USER_ views show information on objects owned by the user, the ALL_ views show information on all the objects accessible by the user, and the DBA_ views show information on all objects in the database. Data dictionary views are available on every type of object in the database, including indexes, constraints, tables, views, synonyms, sequences, and triggers. Additionally, information is available to help the user understand which columns are available in indexes or primary-key constraints. Several views exist to show the position of columns in composite indexes, which are indexes that contain several columns.

Manipulating Oracle Data

There are three types of data change statements available in the Oracle database. They are **update, insert**, and **delete**. The **update** statement allows the user to change row data that already exists in the database. The **insert** statement allows the user to add new row data records to the tables of a database. The **delete** statement allows the user to remove records from the database.

The Importance of Transaction Controls

The various data change operations are supported in Oracle with the usage of transaction-processing controls. There are several different aspects to transaction processing. These include the commands used to set the beginning, middle, and end of transactions, rollback segments designed to store uncommitted data changes, and the locking mechanisms that allow one and only one user at a time to make changes to the data in the database.

Table and Constraint Modifications

There are several activities a developer or DBA can do in order to alter tables and constraints. Some of these activities include adding columns or constraints, modifying the datatypes of columns, or removing constraints.

Adding and modifying columns is accomplished with the **alter table** command, as are adding or modifying constraints on the table. There are several restricting factors on adding constraints, centering around the fact that adding a constraint to a column means that the data already in the column must conform to the constraint being placed upon it.

With respect to adding columns or changing the datatype of a column, there are some general rules to remember. It is easier to increase the size of a datatype for a column, and to add columns to the table. More difficult is changing the datatype of a column from one thing to another. Generally, the column whose datatype is being altered must have NULL values for that column specified for all rows in the table. A table can be dropped with the **drop table** statement. Once dropped, all associated database objects like triggers and constraints, and indexes automatically created to support the constraints, are dropped as well. Indexes that were manually generated by the DBA to improve performance on the table will also be dropped.

There are several other tricks to table alteration. If the user wants to delete all data from a table but leave the definition of the table intact, the user can use the **alter table truncate** command. A database object can be renamed with use of the **rename** command. Alternatively, the DBA can create a synonym, which allows users to reference the database object using a different name. One final option offered to the DBA is to make notes in the database about objects by adding comments. Comments are added with the **comment on** statement.

Sequences

Creation of sequences is another important area of advanced Oracle object creation. A sequence is an object that produces integers on demand according to rules that are defined for the sequence at sequence creation time. Some uses for a sequence include using a sequence to generate primary keys for a table or to generate random numbers. Creating a sequence is accomplished with the **create sequence** command in Oracle. To use a sequence, the user must reference two virtual columns in the sequence, known as CURRVAL and NEXTVAL. The CURRVAL column stores the current value generated by the sequence, while referencing NEXTVAL causes the sequence to generate a new number and replace the value in CURRVAL with that new number.

Several rules can be used to govern how sequences generate their numbers. These rules include the first number the sequence should generate, how the sequence should increment, maximum and minimum values, whether values can be recycled, and others. Modifying the rules that govern sequence integer generation is accomplished with the **alter sequence** statement, while removal of the sequence is accomplished with the **drop sequence** statement.

Views

Views are used to distill data from a table that may be inappropriate for use by some users. Other uses for views include the creation of views that mask the complexity of certain data (such as joins from multiple tables), data that has single-row operations performed on it, and other things. One common example of view usage is the data dictionary, which stores all data about the Oracle database in tables but disallows direct access to the tables in favor of providing views through which the user can select data. There are two categories of views, simple and complex. A simple view is one that draws data from only one table. A complex view is one that draws data from two or more tables. Simple views sometimes allow the user to insert, update, or delete data from the underlying table, while complex views never allow this to occur. A view can also have the option of enforcing a check on the data being inserted. This means that if the user tries to make a change, insertion, or deletion to the underlying table, the view will not allow it unless the view can then select the row being changed. Modifying the definition of a view requires dropping the old view and re-creating it or, alternately, creating the view again with the **or replace** option. The **alter view** statement is used for recompiling an existing view due to a problem with the object dependencies of the database. Removing a view from the database is done with the **drop view** statement.

Indexes

There are several indexes created automatically to support enforcement of uniqueness constraints such as the primary key or the unique constraint. However, the DBA can also create nonunique indexes to support performance improvements on the database application. The traditional index consists of a binary search tree structure. The search algorithm supported by this structure operates by dividing a sorted list of elements in half and comparing the value at hand to the midpoint value, then searching the greater or lesser half depending on whether the value at hand is greater or less than the midpoint value. This process is repeated until the index values are exhausted or the value is found. Studies have shown that this algorithm can find a value from a list of one million elements in 20 or fewer tries. In order for a column to be indexed and used effectively using the B-tree index, the cardinality, or number of distinct values in the column, should be high. To change storage parameters about the index, the DBA can issue the **alter index** statement. To change the number of columns in an index, the index must be dropped and rebuilt. To drop an index, use the **drop** index statement.

Another index available in Oracle is the bitmap index. This index stores each ROWID in the table along with a series of bits, one for every distinct value in the column. The values that are

not used in the column are set off, while the value that is present in the column is set on. Bitmap indexes work well for improving performance on columns with few distinct values.

User Access Control

The Oracle database security model contains three major areas-user authentication, system privileges to control the creation of database objects, and object privileges to control usage of database objects. To change a password, the user can issue the **alter user identified by** statement, specifying the person's username and the desired password. System privileges govern the creation of new database objects, such as tables, sequences, triggers, and views, as well as the execution of certain commands for analyzing and auditing database objects. Three general object maintenance activities are governed by system privileges, and they are the creation, change, and dropping of database objects. Object privileges govern access to an object once it is created, such as selects, updates, inserts, and deletes on tables, execution of packages or procedures, and reference of columns on tables for foreign key constraints.

In situations where there are many users and many privileges governing database usage, the management of privilege granting to users can be improved using roles. Roles act as "virtual users" of the database system. The DBA first defines the privileges a user may need, groups them logically by function or job description, then creates an appropriate role. Privileges to support the function or the job description are then granted to the role, and the role is granted to the user. Roles help to alleviate the necessity of granting several privileges each time a user is added to an application.

Using Synonyms for Database Transparency

Database objects are owned by users and accessible to their schema only, unless permission is explicitly granted by the owner to another user to view the data in the table. Even then, the schema owning the object must be referenced in the statement the user issues to reference the object. Public synonyms can eliminate that requirement, making the schema ownership of the database object transparent. A public synonym is created with the **create public synonym** statement.

Overview of PL/SQL

PL/SQL is the best method available for writing and managing stored procedures that work with Oracle data. PL/SQL code consists of three subblocks-the declaration section, the executable section, and the exception handler. In addition, PL/SQL can be used in four different programming constructs. The types are procedures and functions, packages, and triggers. Procedures and functions are similar in that they both contain a series of instructions that PL/SQL will execute. However, the main difference is that a function will always return one and only one value. Procedures can return more than that number as output parameters. Packages are collected libraries of PL/SQL procedures and functions that have an interface to tell others what procedures and functions are available as well as their parameters, and the body contains the actual code executed by those procedures and functions. Triggers are special PL/SQL blocks that execute when a triggering event occurs. Events that fire triggers include any SQL statement.

Declaring and Using Variables

The declaration section allows for the declaration of variables and constants. A variable can have either a simple or "scalar" datatype, such as NUMBER or VARCHAR2. Alternately, a variable can have a referential datatype that uses reference to a table column to derive its datatype. Constants can be declared in the declaration section in the same way as variables, but with the addition of a constant keyword and with a value assigned. If a value is not assigned to a **constant** in the declaration section, an error will occur. In the executable

section, a variable can have a value assigned to it at any point using the assignment expression (:=).

Using Implicit Cursor Attributes

Using PL/SQL allows the developer to produce code that integrates seamlessly with access to the Oracle database. There are no special characters or keywords required for "embedding" SQL statements into PL/SQL, because SQL is an extension of PL/SQL. As such, there really is no embedding at all. Every SQL statement executes in a cursor. When a cursor is not named, it is called an implicit cursor. PL/SQL allows the developer to investigate certain return status features in conjunction with the implicit cursors that run.

These implicit cursor attributes include **%notfound** and **%found** to identify if records were found or not found by the SQL statement; **%notfound**, which tells the developer how many rows were processed by the statement; and **%isopen**, which determines if the cursor is open and active in the database.

Conditional Statements and Process Flow

Conditional process control is made possible in PL/SQL with the use of **if-then-else** statements. The **if** statement uses a Boolean logic comparison to evaluate whether to execute the series of statements after the **then** clause. If the comparison evaluates to TRUE, the **then** clause is executed. If it evaluates to FALSE, then the code in the **else** statement is executed. Nested **if** statements can be placed in the **else** clause of an **if** statement, allowing for the development of code blocks that handle a number of different cases or situations.

Using Loops

Process flow can be controlled in PL/SQL with the use of loops as well. There are several different types of loops, from simple **loop-exit** statements to **loop-exit when** statements, **while loop** statements, and **for loop** statements. A simple **loop-exit** statement consists of the **loop** and **end loop** keywords enclosing the statements that will be executed repeatedly, with a special **if-then** statement designed to identify if an **exit** condition has been reached. The **if-then** statement can be eliminated by using an **exit when** statement to identify the **exit** condition. The entire process of identifying the **exit** condition as part of the steps executed in the loop can be eliminated with the use of a **while loop** statement. The **exit** condition is identified in the **while** clause of the statement. Finally, the **for loop** statement can be used in cases where the developer wants the code executing repeatedly for a specified number of times.

Explicit Cursor Handling

Cursor manipulation is useful for situations where a certain operation must be performed on each row returned from a query. A cursor is simply an address in memory where a SQL statement executes. A cursor can be explicitly named with the use of the **cursor** *cursor_name* **is** statement, followed by the SQL statement that will comprise the cursor. The cursor cursor_name is statement is used to define the cursor in the declaration section only. Once declared, the cursor must be opened, parsed, and executed before its rows can be manipulated. This process is executed with the **open** statement. Once the cursor is declared and opened, rows from the resultant dataset can be obtained if the SQL statement defining the cursor was a **select** using the **fetch** statement. Both loose variables for each column's value or a PL/SQL record may be used to store fetched values from a cursor for manipulation in the statement.

CURSOR FOR Loops

Executing each of the operations associated with cursor manipulation can be simplified in situations where the user will be looping through the cursor results using the **cursor for** loop statement. The **cursor for** loops handle many aspects of cursor manipulation explicitly. These steps include including opening, parsing, and executing the cursor statement, fetching the value from the statement, handling the exit when data not found condition, and even implicitly declaring the appropriate record type for a variable identified by the loop in which to store the fetched values from the query.

Error Handling

The exception handler is arguably the finest feature PL/SQL offers. In it, the developer can handle certain types of predefined exceptions without explicitly coding error-handling routines. The developer can also associate user-defined exceptions with standard Oracle errors, thereby eliminating the coding of an error check in the executable section. This step requires defining the exception using the **exception_init** pragma and coding a routine that handles the error when it occurs in the exception handler.

For completely user-defined errors that do not raise Oracle errors, the user can declare an exception and code a programmatic check in the execution section of the PL/SQL block, followed by some routine to execute when the error occurs in the exception handler. A special predefined exception called **others** can be coded into the exception handler as well to function as a catchall for any exception that occurs that has no exception-handling process defined. Once an exception is raised, control passes from the execution section of the block to the exception handler. Once the exception handler has completed, control is passed to the process that called the PL/SQL block.