Oracle Database 12c

1. Oracle Database 12c provides support for the \_\_\_\_\_ and \_\_\_\_\_ database models.

Relational, object relational

1. Oracle SQL is a procedural query language.

False

1. The acronym DDL stands for which type of Oracle SQL command?

Data Definition Language

1. The alternative to quoting strings with single quotes is double quotes.

False

1. What single-row function would be most appropriate to use to convert a DATE column to the format "YYYY/MM/DD HH:MI:SS" for a report?

To\_Char not To\_DateTime

1. You want to roll up totals by department, but only see the totals for two of the departments. To do this, you'll have to use the \_\_\_\_\_ and the \_\_\_\_\_ clauses.

Group by and Having not Group by and Where clauses

1. Your query using set operators has multiple subqueries and you want to change the order of evaluation. What is the best method for changing the evaluation order?

Put parentheses around pairs of queries that you want to run before any other set operators.

1. You're using SQL Developer and running some INSERT and UPDATE statements. What is the easiest way to perform a COMMIT on your work so far?

Click on Commit button on the top of the SQL window

1. A datafile in a \_\_\_\_\_ tablespace is limited to about 64 GB if you have a 16K block size.

SMALLFILE not BIGFILE

1. The \_\_\_\_\_ logical database structure is the most granular in an Oracle Database.

Data block

1. What is the difference between a view starting with V$ and a view starting with GV$?

The GV$ view contains an INST\_ID column that identifies which database instance in the cluster this row is referencing.

1. Blocks storing a(n) \_\_\_\_\_ column does not have nor need a setting for PCTFREE.

CLOB not XML not LONG and not IOT

1. Both \_\_\_\_\_ and \_\_\_\_\_ need data in the UNDO tablespace.

Queries and DML

How to install Oracle 12 c:

First install jdk file from Oracle website

<https://www.oracle.com/technetwork/java/javase/downloads/jdk10-downloads-4416644.html>

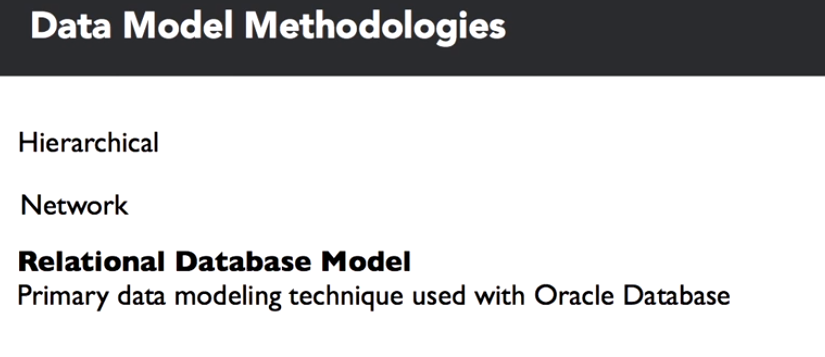
Install SQL Developer from <https://goo.gl/nftWBA> -> Redirects to

<https://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>

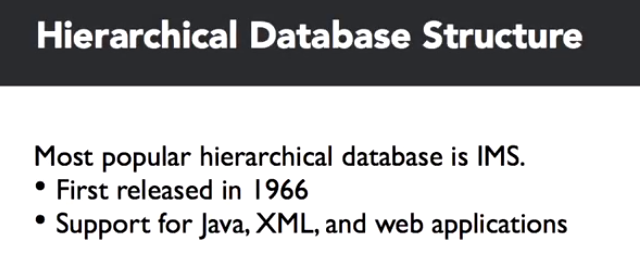
what you should know:

* Access to an Oracle 12 c database
* Install SQL Developer (free) -link above
* Basic understanding of relational databases
* Some experience with a dialect of SQL
* Understanding of client-server architecture and networks
* Database application development experience helpful

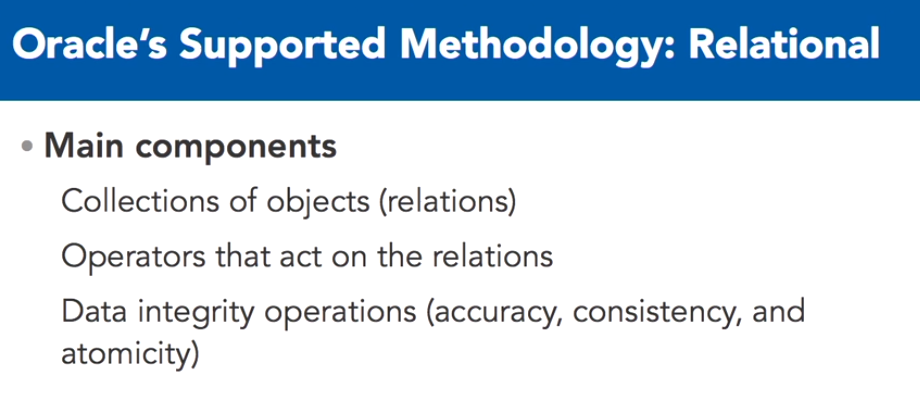
Whether you're using a relational database, hierarchical, network, or an object-relational database, you need to use a data modeling tool to document the objects, attributes and relationships between the objects in the database. In the '60s and '70s, both hierarchical and network data models were more prevalent, mostly because of limitations with the hardware and software available at the time. As computing power and capabilities increased, other models emerged such as the relational database model. The primary data modeling technique used with Oracle Database is the relational model.



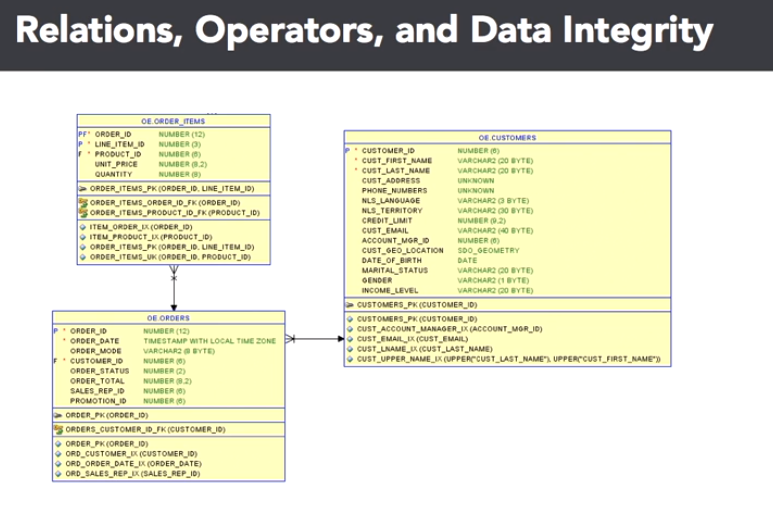
The most popular hierarchical database, IMS, was first released in 1966 by IBM in conjunction with Rockwell and Caterpillar as a database to support bill of materials, application management for the Apollo space program. IMS has evolved enough to be a viable platform well into the 21st century with support for Java, XML and web applications. IDMS, Integrated Database Management System, is an example of the network database model, first developed at BF Goodrich in 1973 and later marketed commercially by Cullinet.



Relational database modeling was born in the early 1970's with a paper by Dr. E. F. Codd in June of 1970, entitled, "A Relational Model of Data for Large Shared Data Banks." So, what is a relational model? It's really three things: a collection of objects, relations, operators that act on the relations, and data integrity operations.

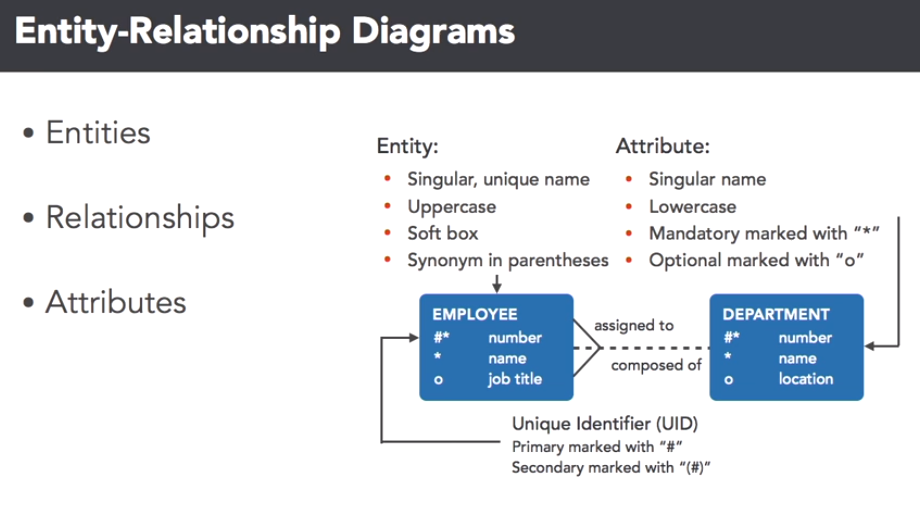


The collection of objects represent real world things such as a list of auto parts, a balance sheet, or in this case, customer orders, much like a spreadsheet.

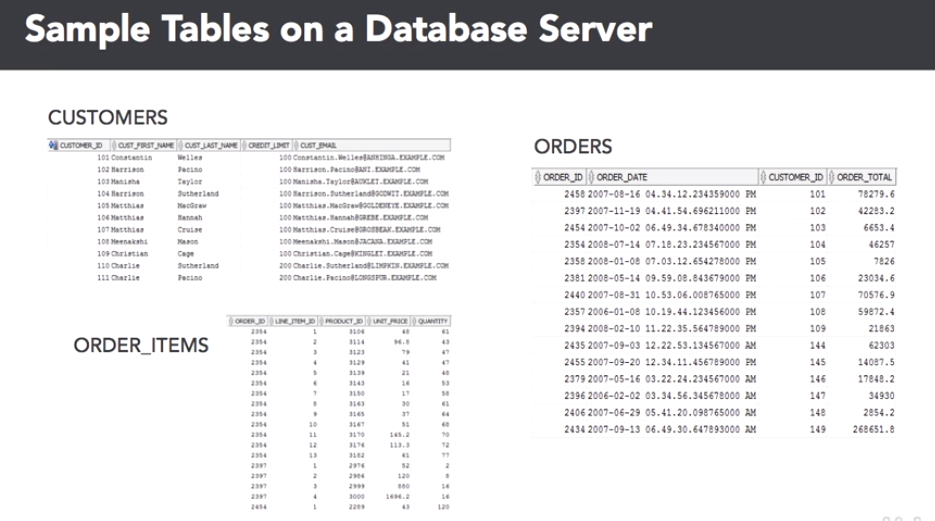


These are what are called relations. Second, a relational model has a set of operators that act on one or more relations. These operators include filtering the components of a relation to get the desired results. In a spreadsheet, the original version of a database table, you might use a filter operation to only view certain columns, or you might only want to see certain rows of the spreadsheet. Finally, you must have one of the most important components of the relational model: data integrity operations.

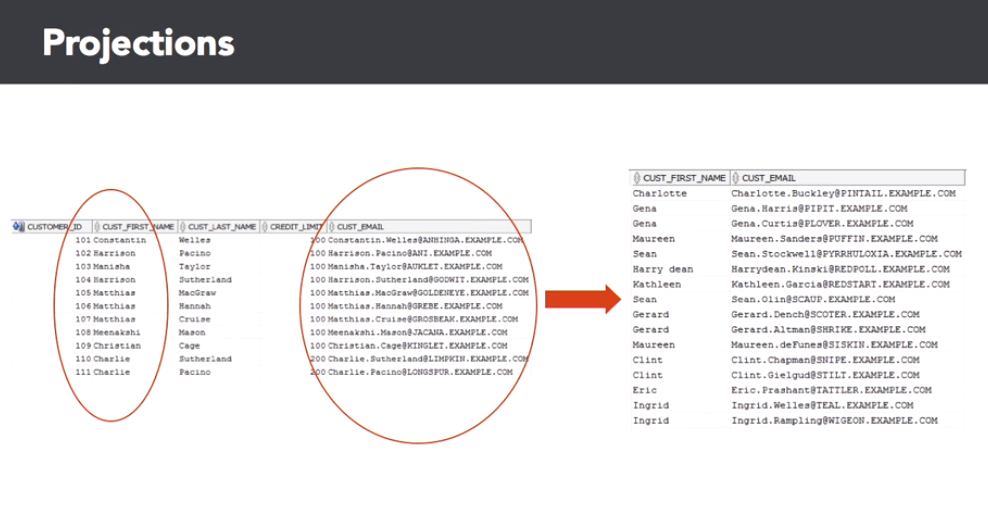
For subcomponents of any given relation, you want to make sure that the information in the relation is accurate and logically consistent with the information in the other relations of your data model. We'll point out these components in the following slides. You've probably noticed some interesting nomenclature and symbols in the previous slide. That is the logical implementation of those business entities in Oracle Sequel Data Modeler, of an entity relationship diagram. These diagrams make it easy to identify relations among the entities along with the business rules that apply to the attributes in each entity.



We'll use this notation throughout this course when referencing logical data model components. Here, we see a small subset of tables in the database.

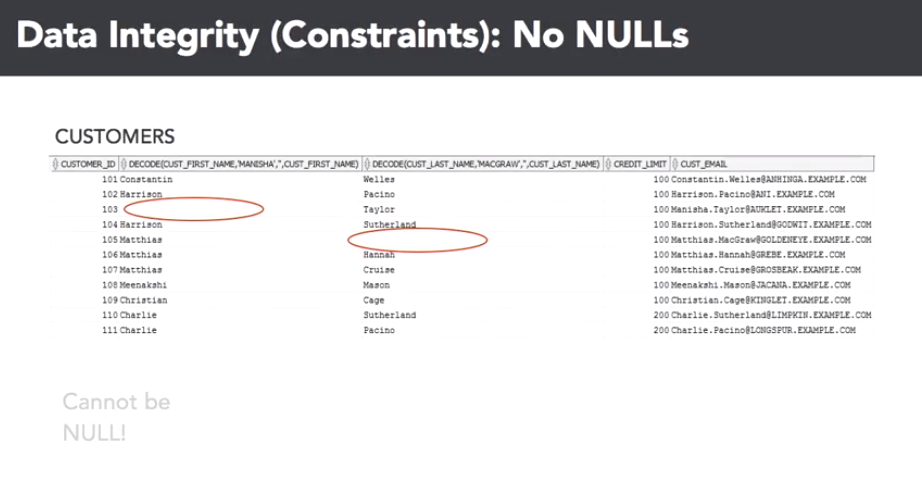


We'll get back to the data modeling development part in a bit. The three relations stored as tables in the database are the customers, orders and order item tables. Notice that there are many operators you can use on these tables. For example, you may use a projection on the customer's table to only see a customer's first name and e-mail address or you may want to see all information about a single customer's orders using a selection on the order items table.

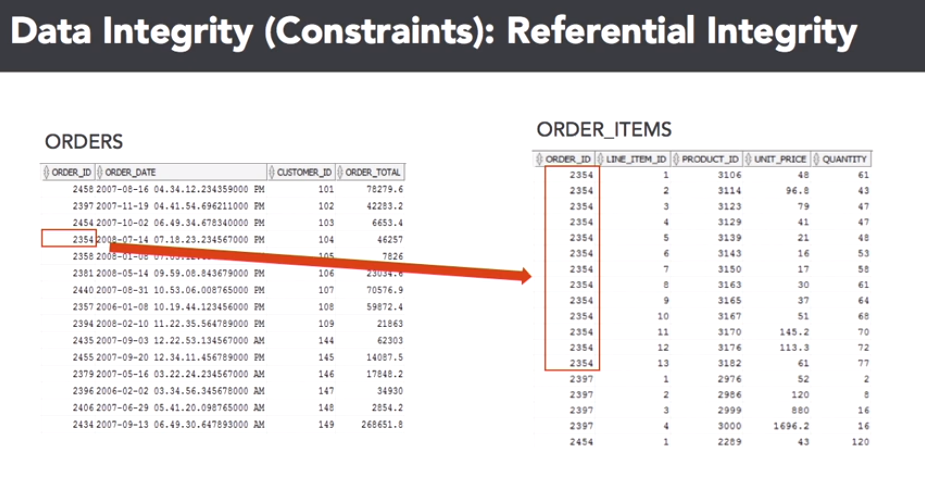


Finally, you may want to use a join operator on the orders and order items table to generate a new relation that shows a given order number and its associated line items.

Let's talk about data integrity next. When defining a relation, you may also want to define the data integrity operations that ensure data quality and consistency. For example, you don't want to have any missing data for custfirst name or custlast name in the customers table, as you can see in the slide.



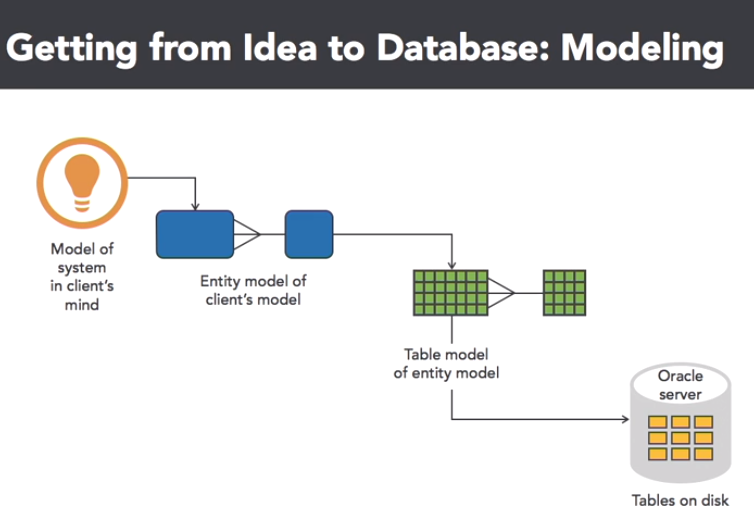
Also, you want to make sure that the order ID column in the order items table has an entry in the orders table. This is called a foreign key constraint.



How do you get from an idea in a CEO's head to the actual data model? Well, when you want to create an Oracle Database for an application to fulfill a business need, you usually don't want to start creating tables on a new database as your first step. It's important to start with the application stakeholders to gather business requirements for the data model. Frequently, the application owners have an idea in their head of the types of data they want to store, but that doesn't translate directly to Oracle Database tables.

After the business requirements are gathered, you'll facilitate the creation of an entity relationship, ER diagram, to capture the entities, relations and their attributes, which are usually columns, that are important to the application along with the relationships between the entities and the data integrity constraints. This would be considered the logical ER diagram. From the logical ER diagram, the modeler will generate the database table model for the diagram, in other words, the Oracle Database statements that will be executed to create the physical representations of the entities, attributes and relationships, not to be confused with relations, defined in the ER diagram.

Finally, the Oracle DBA, which may also be you, will run the commands to create the first version of the database tables for the application.

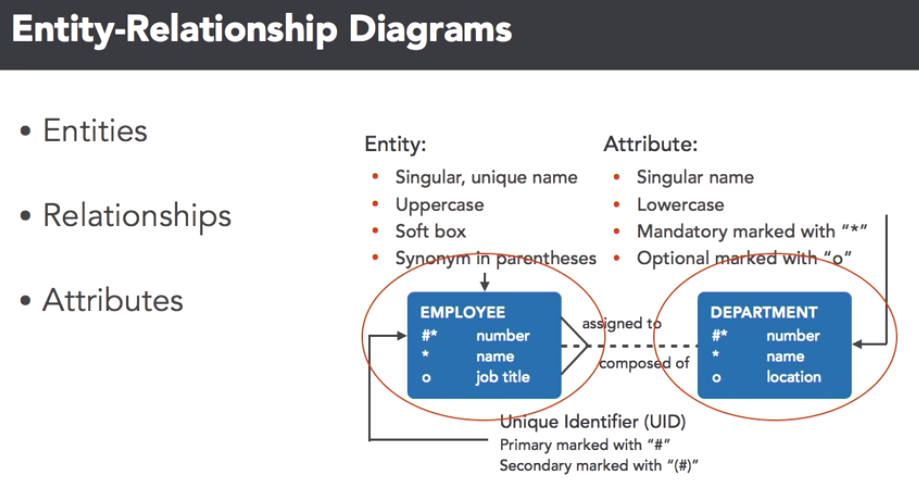


So, what do we know now? Several things. Oracle Database is primarily a relational database platform with some object-oriented capabilities thrown in, and is firmly based on the relational database model, first proposed by E. F. Codd in 1970. An entity relationship ER diagram shows the logical objects and relationships in a relational database.

These objects will map almost one to one with the physical objects in the database itself.

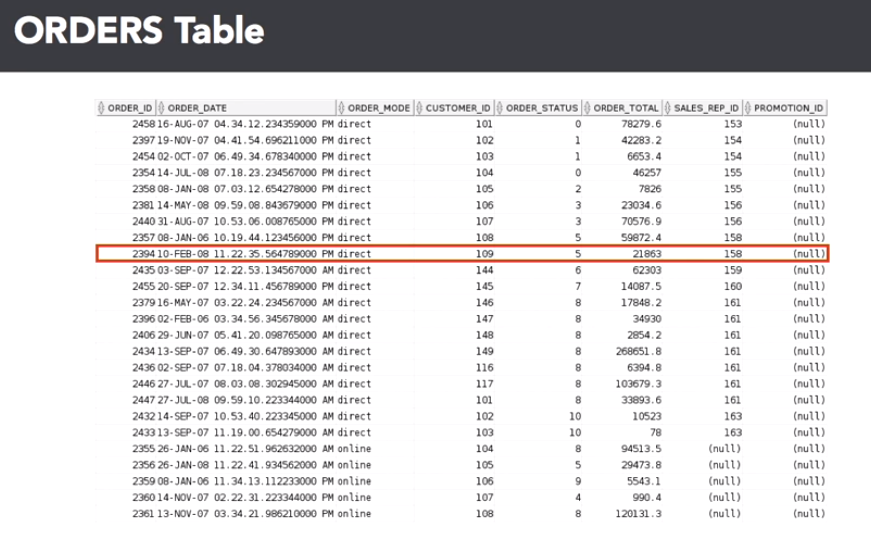
Logical database modeling

Let's start out this lesson by analyzing the components in a logical data model diagram.



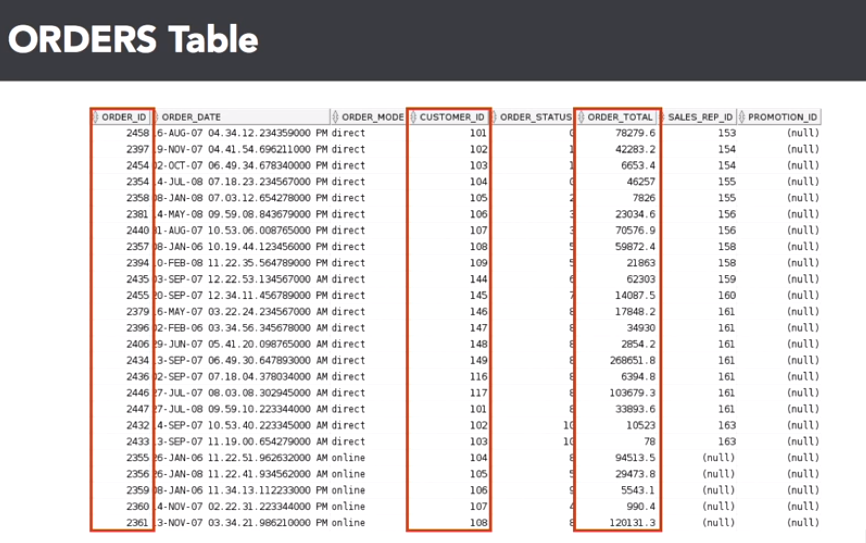
At this level, you're not as concerned about data types, but instead, what elements of your organization are important from a business perspective. Each blue box is one of the entities, remember, this is also called a relation and data modeling terminology, in this case, employees and departments. You'll use certain conventions in your data modeling tool, to assign characteristics to the attributes of each entity.

The entity name is in upper case, the attributes are in lower case, and both are almost always singular. You'll also want to mark certain attributes as being unique, or not knowable, to enforce data integrity. Here, the employee number, and department number, are unique identifiers, and the name attribute in both of those entities, can't be empty. No worries, a lot of these requirements are easy to implement, in most data modeling tools, as you'll see later in this video. In your favorite data modeling tool, you'll also define the relationships between the entities.

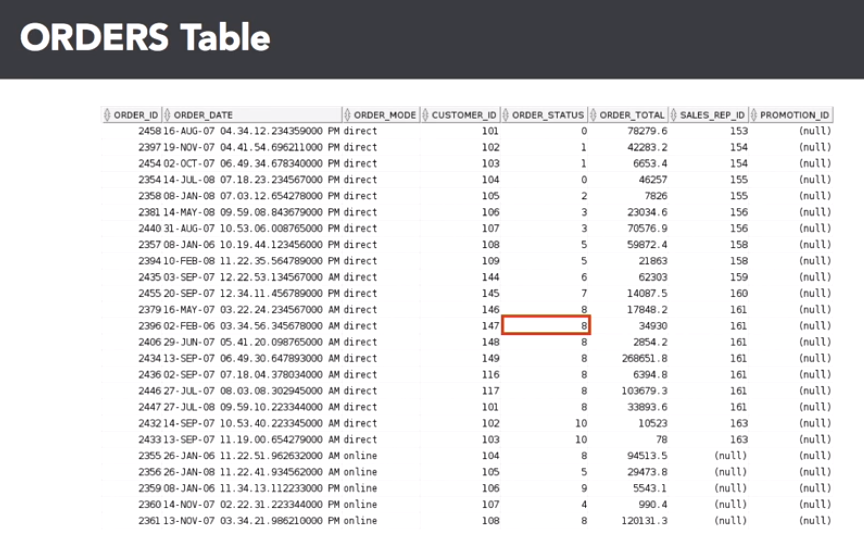


This is a critical step to ensure that data integrity will be enforced using database features such as foreign keys, as well as defining the cardinality of the entities in your database. After you've created the table in the Oracle database, it will look something like this in SQL Developer. Let's review the connection between the elements in your data model diagram, with the database table itself. Here are the first 25 rows of the order table. A row in the database, is a tuple in a classic relational database model.

And you select rows using relational database languages, such as a SQL SELECT statement structured query language. In this diagram, the row with an order ID of 2394 is highlighted. A column, also known as a projection, is the attribute of an entity. A column can have several attributes beyond what kind of data type it is. It can be, one, a unique identifier, two, an identifier that links to a unique column in another table.

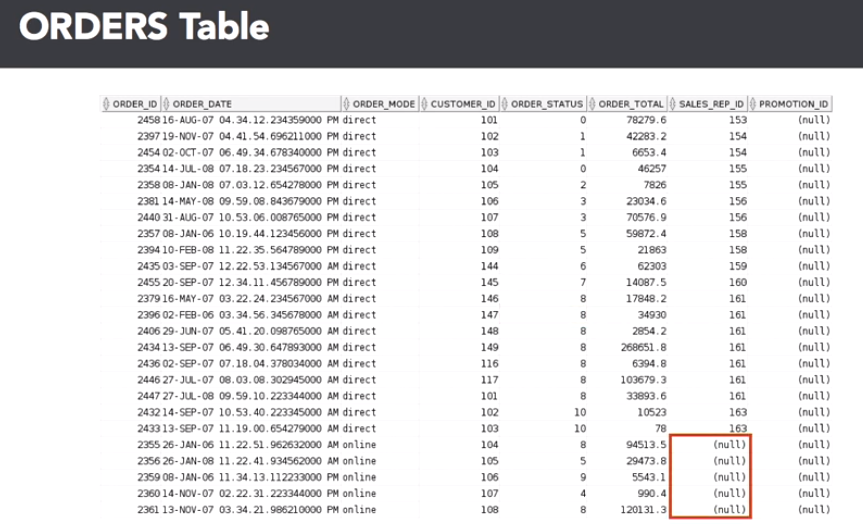


Or three, just to another column, usually dependent on the unique identifier in that row. An intersection of a row and column is a field, a single atomic value of a column, in a single database row.

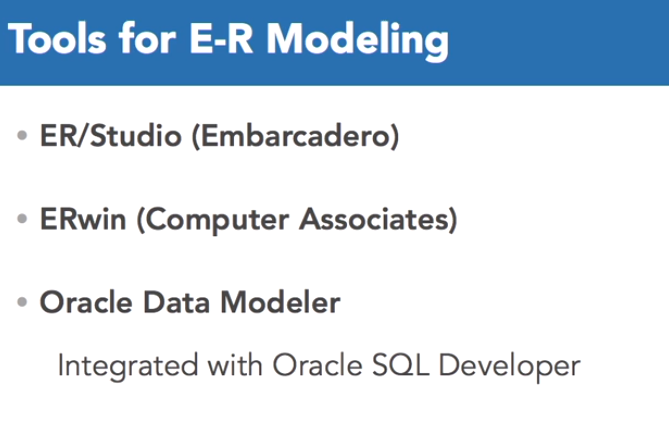


The highlighted field answers the question, "What is the order status for order number 2396?" For a customer service representative that is inquiring on the order status, seeing a value of eight might not be very useful or obvious, therefore the result of this query will probably be joined to the order status table, to get the text description.

Finally, any non-primary key columns can have a value of null, meaning that the field has no value, or it's unknown. It does not mean that it is blank, or has spaces, or has a value of zero, it does not exist. This has many implications when selecting rows, or joining tables, based on the value, or lack thereof, in this column.



Now that you know what you want to do, which is to create the data model, you can use one of many good tools from vendors such as Embarcadero, and Computer Associates, to perform your data modeling tasks.



A big advantage to using Oracle Data Modeler, is that it's integrated into Oracle SQL Developer, allowing you to manage the entire database development and deployment life cycle within one gooey, and on top of it all, it's free.

Here's how well-integrated Oracle SQL Developer is with Oracle Data Modeler. Going to the file tab, select data modeler, then select import. As you can see, there are a wide variety of other data sources, in this case we're going to import from data dictionary.

Next, pick a connection to the database you want to use for the import, I'm going to click on the OE connection. Click on next, and you'll see a list of the schemas in the database. I'm going to pick the OE schema, since it contains the order entry tables. Go ahead and click that check box, and then click next. Right here, you'll see all the tables that OE owns, and there are the two we want, orders, and order items. So I'm going to select both of those, and then click on next.

Before performing the import, you'll see a summary screen, click on finish. As the gears grind, it's generating the design for the data model design for you. There's a summary of what it did, how many statements it imported and so forth. You can save the summary of the import operation, or you can close it. In this case, I'm going to close it. What you see is the imported data model, including the foreign key relationship between the orders and order items table. You will also be able to use SQL Developer long after the data model has been deployed, for report writing, ad hoc queries, and DBA specific duties, such as managing users, table spaces, and database server resources.

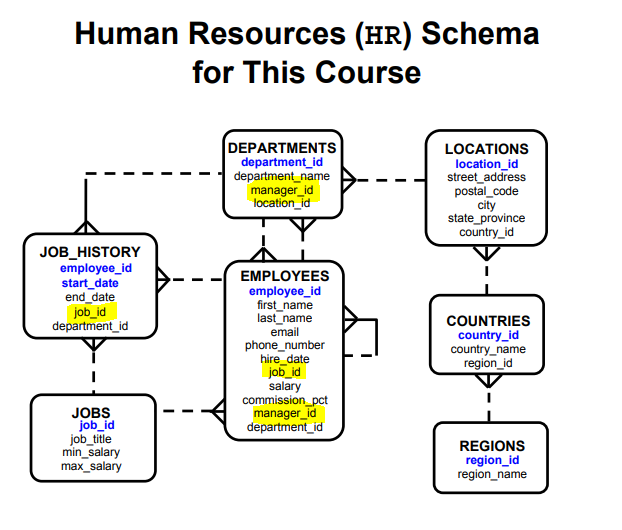
Here's a built-in report to list the columns for all views in the database. An Oracle DBA is only as good, and as efficient, as the tool sets they use. Using Oracle's free data modeling, deployment, and development tools, will help any DBA power user be successful in their job. The tools that Oracle provides can truly be one-stop shopping for all of your database design, development, and maintenance needs.

**PL/SQL**

PL/SQL is a language that has programming features that serve as an extension to SQL. SQL, which is a nonprocedural language, is made procedural with PL/SQL programming constructs.

PL/SQL applications can run on any platform or operating system on which an Oracle server runs. In this lesson, you learned how to build basic PL/SQL blocks.

Human Resources (HR) Schema for This Course



**Human Resources (HR) Schema Description**

The Human Resources (HR) schema is part of the Oracle Sample Schemas that can be installed

in an Oracle database. The practice sessions in this course use data from the HR schema.

**Table Descriptions**

• REGIONS contains rows that represent a region such as Americas, Asia, and so on.

• COUNTRIES contains rows for countries, each of which is associated with a region.

• LOCATIONS contains the specific address of a specific office, warehouse, or production site of a company in a particular country.

• DEPARTMENTS shows details about the departments in which employees work. Each department may have a relationship representing the department manager in the

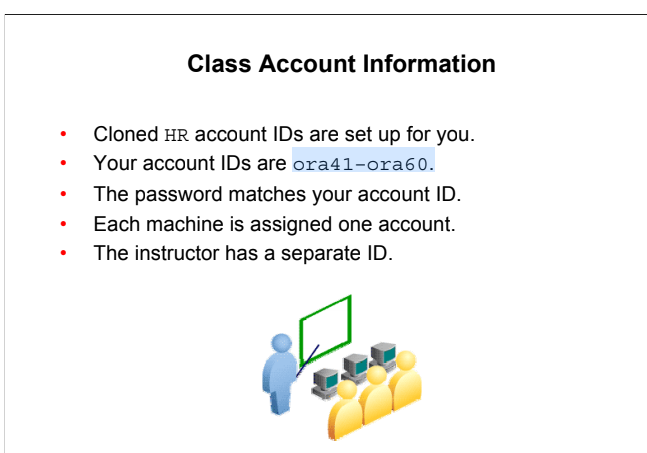
EMPLOYEES table.

• EMPLOYEES contains details about each employee working for a department. Some employees may not be assigned to any department.

• JOBS contains the job types that can be held by each employee.

• JOB\_HISTORY contains the job history of the employees. If an employee changes

departments within a job or changes jobs within a department, then a new row is inserted into this table with the old job information of the employee.



**PL/SQL Development Environments**

Oracle provides several tools that can be used to write PL/SQL code. Some of the development tools that are available for use in this course:

* Oracle SQL Developer: A graphical tool
* Oracle SQL\*Plus: A window or command-line application
* Oracle JDeveloper: A window-based integrated development environment (IDE)

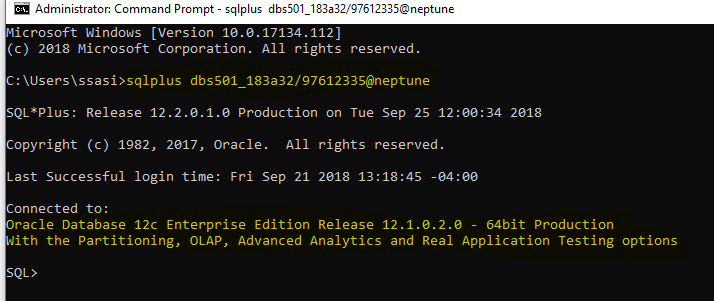
Note: The code and screen examples presented in the course notes were generated from output in the SQL Developer environment.



SQL Developer, the visual tool for database development, simplifies the following tasks:

* Browsing and managing database objects
* Executing SQL statements and scripts
* Editing and debugging PL/SQL statements
* Creating reports

You can connect to any target Oracle database schema using standard Oracle database authentication. When connected, you can perform operations on objects in the database.



**Coding PL/SQL in SQL\*Plus**

Oracle SQL\*Plus is a graphical user interface (GUI) or command-line application that enables you to submit SQL statements and PL/SQL blocks for execution and receive the results in an application or a command window.

SQL\*Plus is:

* Shipped with the database
* Installed on a client and on the database server system
* Accessed from an icon or the command line

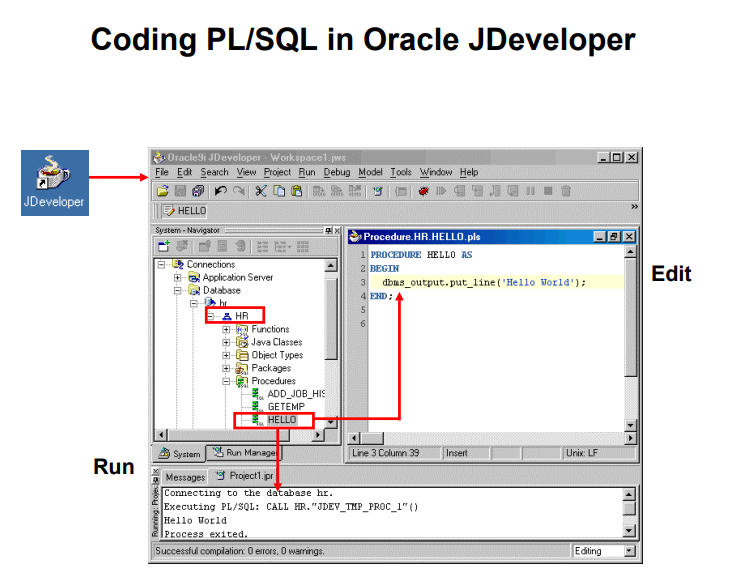
When coding PL/SQL subprograms using SQL\*Plus, remember the following:

* You create subprograms by using the CREATE SQL statement.
* You execute subprograms by using either an anonymous PL/SQL block or the **EXECUTE** command.

• If you use the **DBMS\_OUTPUT** package procedures to print text to the screen, you must first execute the **SET SERVEROUTPUT ON** command in your session.

**Coding PL/SQL in Oracle JDeveloper**

Oracle JDeveloper allows developers to create, edit, test, and debug PL/SQL code by using a sophisticated GUI. Oracle JDeveloper is a part of Oracle Developer Suite and is also available as a separate product.



When coding PL/SQL in JDeveloper, consider the following:

* You first create a database connection to enable JDeveloper to access a database schema owner for the subprograms.
* You can then use the JDeveloper context menus on the Database connection to create a new subprogram construct using the built-in JDeveloper Code Editor. The JDeveloper

Code Editor provides an excellent environment for PL/SQL development, with features such as the following:

* Different colors for syntactical components of the PL/SQL language
* Code insight to rapidly locate procedures and functions in supplied packages

• You invoke a subprogram by using a Run command on the context menu for the named subprogram. The output appears in the JDeveloper Log Message window, as shown in the lower portion of the screenshot.

Note: JDeveloper provides color-coding syntax in the JDeveloper Code Editor and is sensitive to PL/SQL language constructs and statements

**Creating a Database Connection**

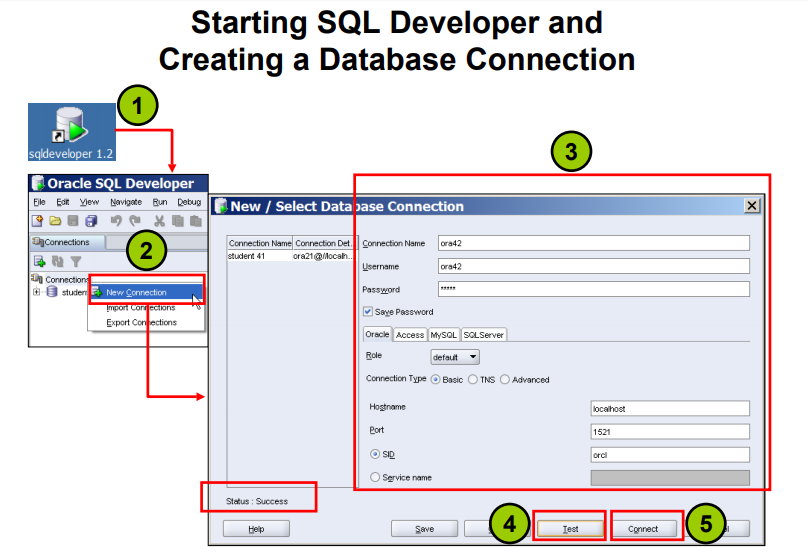
To create a database connection, perform the following steps:

1. Double-click <your\_path>\sqldeveloper\sqldeveloper.exe.
2. On the Connections tabbed page, right-click Connections and select New Connection.
3. Enter the connection name, username, password, host name, and SID for the database you want to connect to.
4. Click Test to make sure that the connection has been set correctly.
5. Click Connect.

On the basic tabbed page, at the bottom, enter the following options:

* **Hostname**: Host system for the Oracle database
* **Port**: Listener port
* **SID**: Database name
* **Service name**: Network service name for a remote database connection

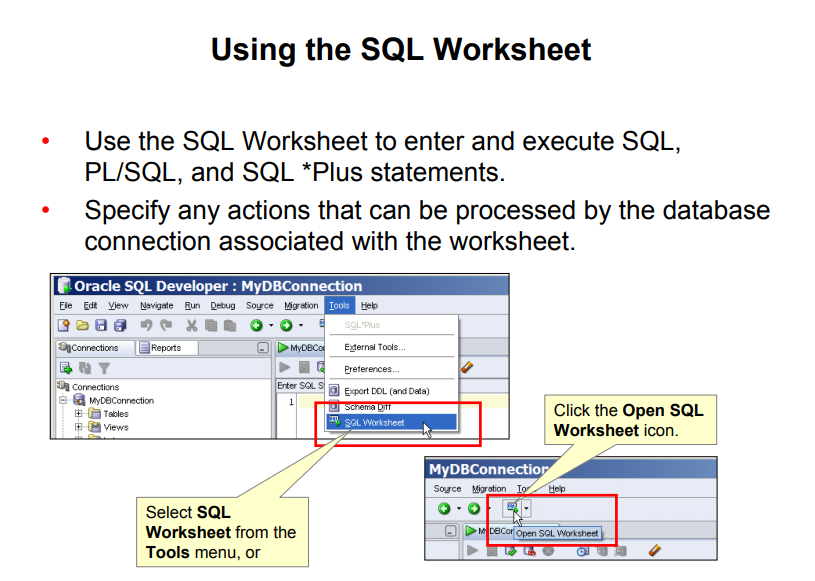
If you select the Save Password check box, the password is saved to an XML file. So, after you close the SQL Developer connection and open it again, you will not be prompted for the password.



**Creating Schema Objects**

* You can create any schema object in SQL Developer
* using one of the following methods:
  + Executing a SQL statement in the SQL Worksheet
  + Using the context menu
* Edit the objects using an edit dialog box or one of the many context-sensitive menus.

• View the DDL for adjustments such as creating a new object or editing an existing schema object.



**Using the SQL Worksheet**

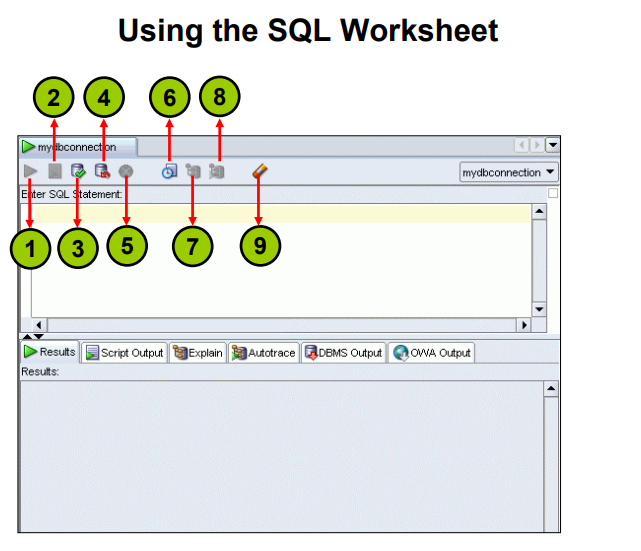
When you connect to a database, a SQL Worksheet window for that connection is automatically opened. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL\*Plus statements. The SQL Worksheet supports SQL\*Plus statements to a certain extent. SQL\*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database.

You can specify any actions that can be processed by the database connection associated with the worksheet, such as:

* Creating a table
* Inserting data
* Creating and editing a trigger
* Selecting data from a table
* Saving the selected data to a file

You can display a SQL Worksheet by using any of the following:

* Select Tools > SQL Worksheet.
* Click the Open SQL Worksheet icon.



You may want to use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of SQL statements that you have executed.

You can use the SQL Worksheet toolbar that contains icons to perform the following tasks:

**1. Execute Statement**: Executes the statement at the cursor in the Enter SQL Statement box.

You can use bind variables in the SQL statements, but not substitution variables.

**2. Run Script**: Executes all statements in the Enter SQL Statement box using the Script

Runner. You can use substitution variables in the SQL statements, but not bind variables.

**3. Commit**: Writes any changes to the database and ends the transaction

**4. Rollback**: Discards any changes to the database, without writing them to the database, and ends the transaction

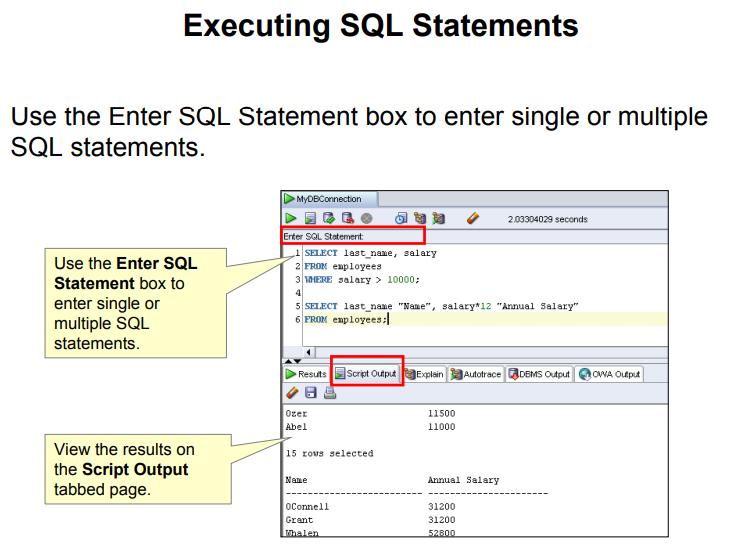
**5. Cancel**: Stops the execution of any statements currently being executed

6. SQL History: Displays a dialog box with information about SQL statements that you have executed

**7. Execute Explain Plan**: Generates the execution plan, which you can see by clicking the Explain tab

**8. Autotrace**: Generates trace information for the statement

**9. Clear:** Erases the statement or statements in the Enter SQL Statement box



**Executing SQL Statements**

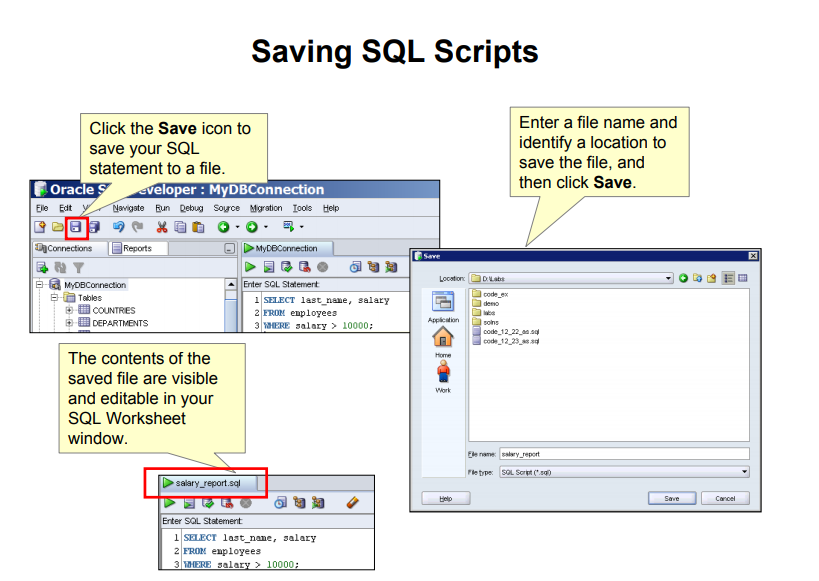
In the SQL Worksheet, you can use the Enter SQL Statement box to enter a single or multiple SQL statements. For a single statement, the semicolon at the end is optional.

When you enter the statement, the SQL keywords are automatically highlighted. To execute a SQL statement, ensure that your cursor is within the statement and click the Execute Statement icon. Alternatively, you can press the **F9** key.

To execute multiple SQL statements and see the results, click the Run Script icon.

Alternatively, you can press the **F5** key.

In the example in the slide, because there are multiple SQL statements, the first statement is terminated with a semicolon. The cursor is in the first statement and, therefore, when the statement is executed, results corresponding to the first statement are displayed in the Results box.



**Saving SQL Scripts**

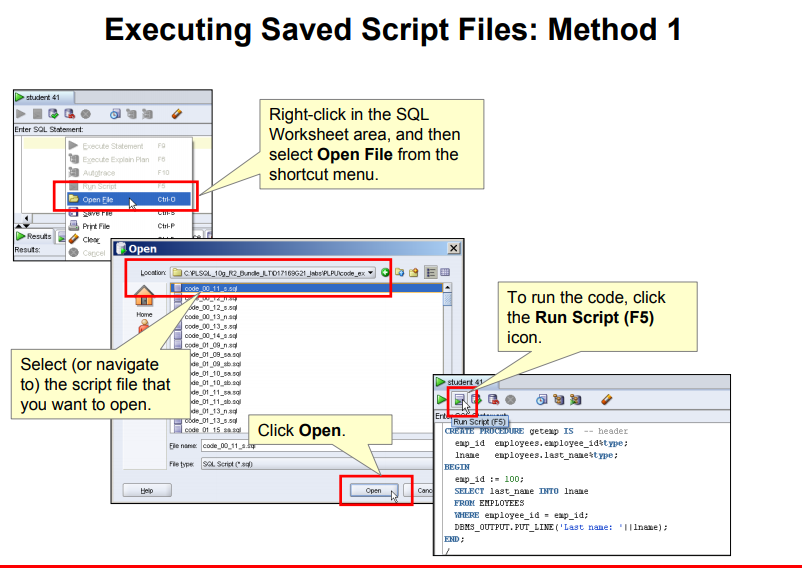
You can save your SQL statements from the SQL Worksheet into a text file. To save the contents of the Enter SQL Statement box, follow these steps:

1. Click the Save icon or use the File > Save menu item.
2. In the Windows Save dialog box, enter a file name and the location where you want the file saved.
3. Click Save.

After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at once. Each file displays as a tabbed page.

**Script Pathing**

You can select a default path to look for scripts and to save scripts. Under **Tools > Preferences > Database > Worksheet Parameters**, enter a value in the **Select default path to look for** **scripts** field.



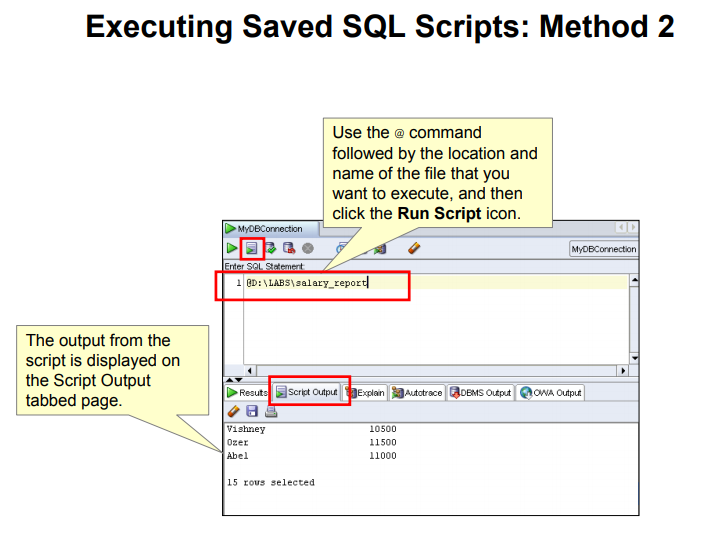
**Executing Saved Script Files: Method 1**

You can open and execute a script file in the SQL Worksheet area as follows:

1. Right-click in the SQL Worksheet area, and then select Open File from the pop-up menu.

The Open dialog box is displayed.

1. In the Open dialog box, select (or navigate to) the script file that you want to open.
2. Click Open. The code of the script file is displayed in the SQL Worksheet area.
3. To run the code, click Run Script (F5) on the SQL Worksheet toolbar.



**Executing Saved Script Files: Method 2**

To run a saved SQL script, perform the following steps:

1. In the Enter SQL Statement window, use the @ command, followed by the location and name of the file that you want to run.
2. Click the Run Script icon.

The results from running the file are displayed on the Script Output tabbed page. You can also save the script output by clicking the Save icon on the Script Output tabbed page. The Windows

File Save dialog box appears and you can identify a name and location for your file.

Output improvements:

SET LINESIZE 32000;

SET PAGESIZE 40000;

SET LONG 50000;

SPOOL output.txt

**Practice**

This is the first of many practices in this course. The solutions (if you require them) can be found in Appendix A. Practices are intended to cover most of the topics that are presented in the corresponding lesson.

1. Start up SQL Developer using the user ID and password that are provided to you by the instructor such as oraxx where xx is the number assigned to your PC.

2. Create a database connection using the following information:

* 1. Connection Name: MyDBConnection.
  2. Username: oraxx where xx is the number assigned to your PC by the instructor.
  3. Password: oraxx where xx is the number assigned to your PC by the instructor.
  4. Hostname: Enter the host name for your PC.
  5. Port: 1521
  6. SID: ORCL

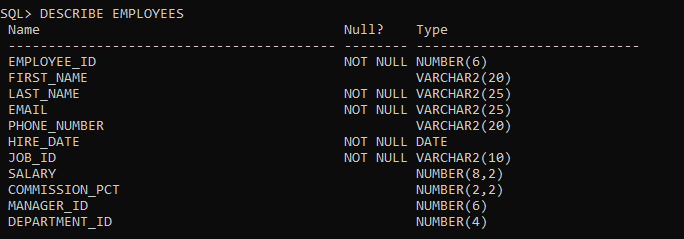
3. Test the new connection. If the Status is Success, connect to the database using this new connection.

1. Double-click the MyDBConnection icon on the Connections tabbed page.
2. Click the Test button in the New/Select Database Connection window. If the status is Success, click the Connect button.

4. Browse the structure of the EMPLOYEES table and display its data.

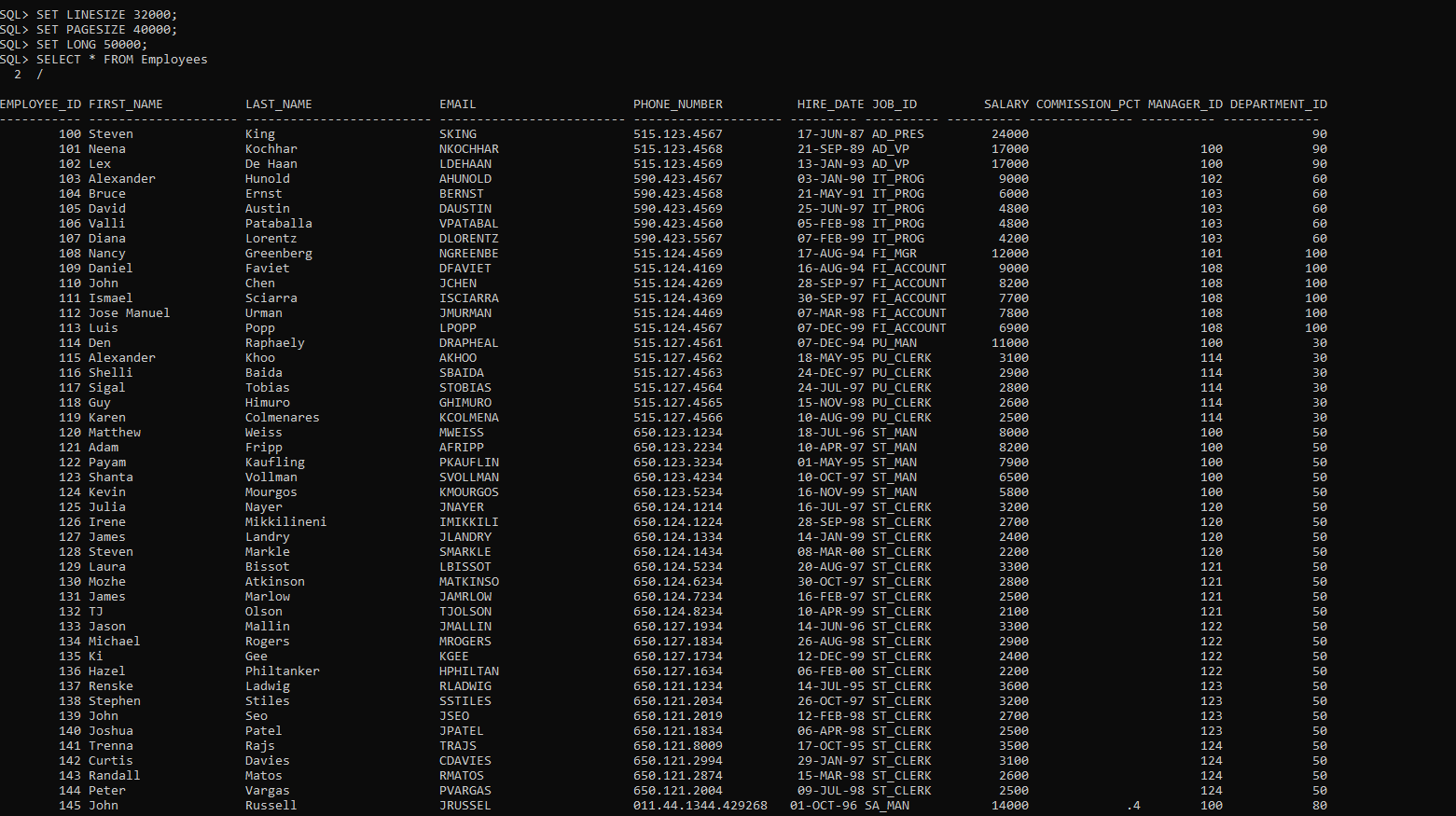
1. Expand the MyDBConnection connection by clicking the plus sign next to it.
2. Expand the Tables icon by clicking the plus sign next to it.
3. Display the structure of the EMPLOYEES table.

Sali >> DESCRIBE Employees



1. Browse the EMPLOYEES table and display its data.

SELECT \* FROM Employees



1. Use the SQL Worksheet to select the last names and salaries of all employees whose annual salary is greater than $10,000. Use both the Execute Statement (F9) and the Run Script icon (F5) icons to execute the SELECT statement. Review the results of both methods of executing the SELECT statements in the appropriate tabs.

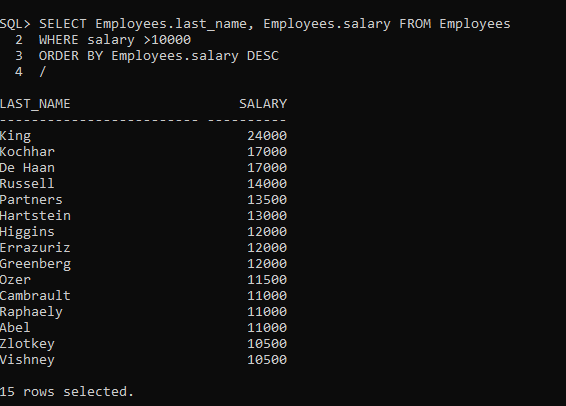
Sali >>

SELECT Employees.last\_name, Employees.salary FROM Employees

WHERE salary >10000

ORDER BY Employees.salary DESC

/



Note: Take a few minutes to familiarize yourself with the data, or consult Appendix B, which provides the description and data for all tables in the HR schema that you will use in this course.

7. In the SQL Developer menu, navigate to **Tools > Preferences**. The Preferences window is displayed.

8. Click the Worksheet Parameters option under the Database option. In the “Select default path to look for scripts” text box, specify the D:\labs\PLSF folder. This folder contains the solutions scripts, code examples scripts, and any labs or demos used in this course.

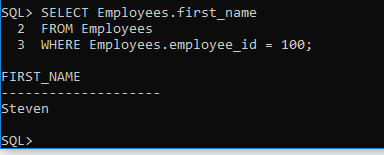
Oracle University and ORACLE CORPORATION use only THESE eKIT MATERIALS ARE FOR YOUR USE.

Sali >>>

SELECT Employees.first\_name

FROM Employees

WHERE Employees.employee\_id = 100;



Sali >>

SET SERVEROUTPUT ON;

DECLARE

v\_name VARCHAR(20);

BEGIN

SELECT Employees.first\_name INTO v\_name

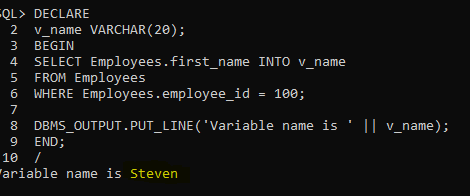
FROM Employees

WHERE Employees.employee\_id = 100;

DBMS\_OUTPUT.PUT\_LINE('Variable name is ' || v\_name);

END;

/



PL/SQL is not case sensitive.

V\_name is equal to V\_NaME ….

Sali >>

DECLARE

V\_naMe VARCHAR(20);

BEGIN

SELECT Employees.first\_name INTO v\_name

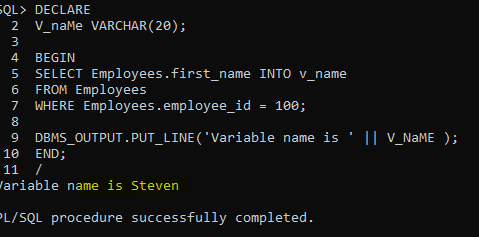
FROM Employees

WHERE Employees.employee\_id = 100;

DBMS\_OUTPUT.PUT\_LINE('Variable name is ' || V\_NaME );

END;

/



Writing the “ Employees.employee\_id “ is not a must

DECLARE

V\_naMe VARCHAR(20);

BEGIN

SELECT Employees.first\_name INTO v\_name

FROM Employees

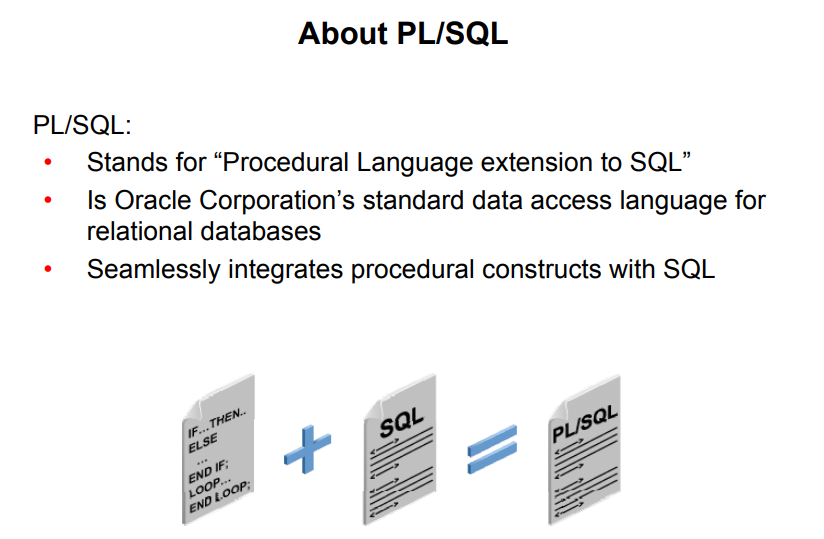
WHERE employee\_id = 100;

DBMS\_OUTPUT.PUT\_LINE('Variable name is ' || V\_NaME );

END;

/

**About PL/SQL**

****

Structured Query Language (SQL) is the primary language used to access and modify data in relational databases. There are only a few SQL commands, so you can easily learn and use them.

Consider an example:

SELECT first\_name, department\_id, salary FROM employees;

The SQL statement shown above is simple and straightforward. However, if you want to alter any data that is retrieved in a conditional manner, you soon encounter the limitations of SQL.

Consider a slightly modified problem statement: For every employee retrieved, check the department ID and the salary. Depending on the department’s performance and also the employee’s salary, you may want to provide varying bonuses to the employees.

Looking at the problem, you know that you have to execute the preceding SQL statement, collect the data, and apply logic to the data. One solution is to write a SQL statement for each department to give bonuses to the employees in that department. Remember that you also have to check the salary component before deciding the bonus amount. This makes it a little complicated. You now feel that it would be much easier if you had conditional statements.

PL/SQL is designed to meet such requirements. It provides a programming extension to the already-existing SQL.

PL/SQL:

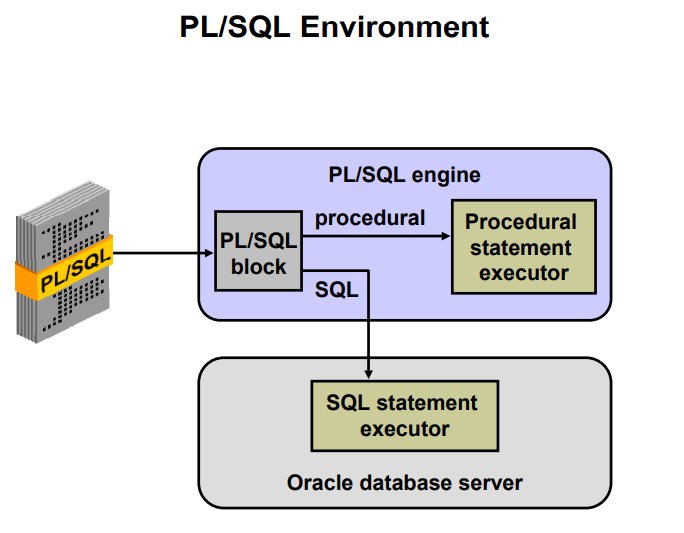
• Provides a block structure for executable units of code.

Maintenance of code is made easier with such a well-defined structure.

• Provides procedural constructs such as:

* + Variables, constants, and data types
  + Control structures such as conditional statements and loops
  + Reusable program units that are written once and executed many times

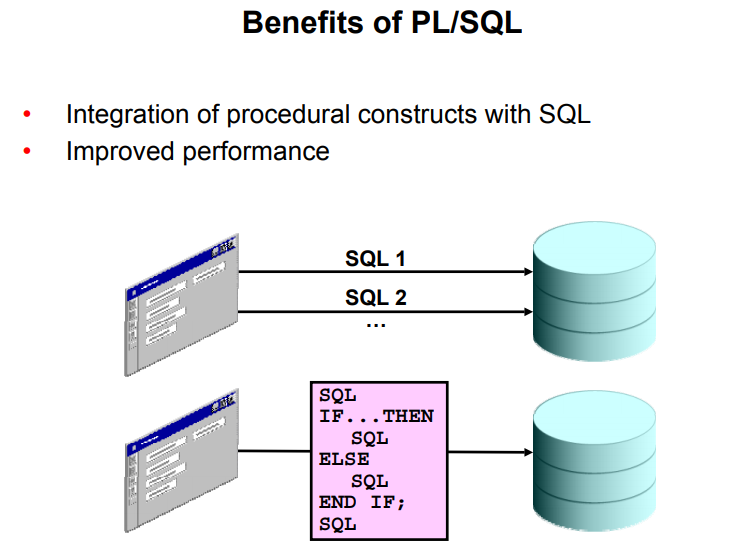
PL/SQL offers modern software engineering features such as data encapsulation, exception handling, information hiding, and object orientation. It brings state-of-the-art programming to the Oracle server and toolset. PL/SQL provides all the procedural constructs that are available in any third-generation language (3GL).



**PL/SQL Environment**

The slide shows the PL/SQL execution environment in the Oracle database server. A PL/SQL block contains procedural statements and SQL statements. When you submit the PL/SQL block to the server, the PL/SQL engine first parses the block. The PL/SQL engine identifies the procedural statements and the SQL statements. It passes the procedural statements to the procedural statement executor and the SQL statements to the SQL statement executor individually.

The diagram in the slide shows the PL/SQL engine within the database server. The Oracle application development tools can also contain a PL/SQL engine. The tool passes the blocks to its local PL/SQL engine. Therefore, all procedural statements are executed locally and only the SQL statements are executed in the database. The engine used depends on where the PL/SQL block is being invoked from.

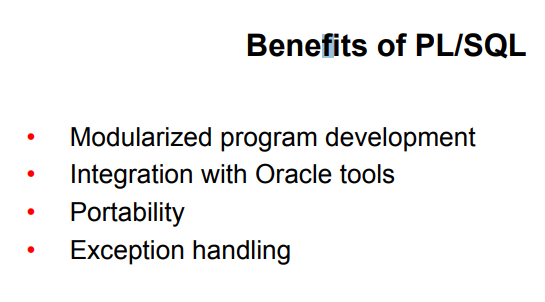


**Benefits of PL/SQL**

Integration of procedural constructs with SQL: The most important advantage of PL/SQL is the integration of procedural constructs with SQL. SQL is a nonprocedural language. When you issue a SQL command, your command tells the database server what to do. However, you cannot specify how to do it. PL/SQL integrates control statements and conditional statements with SQL, giving you better control of your SQL statements and their execution. Earlier in this lesson, you saw an example of the need for such integration.

Improved performance: Without PL/SQL, you would not be able to logically combine SQL statements as one unit. If you have designed an application containing forms, you may have many different forms with fields in each form. When a form submits the data, you may have to execute a number of SQL statements. SQL statements are sent to the database one at a time. This results in many network trips and one call to the database for each SQL statement, thereby increasing network traffic and reducing performance (especially in a client/server model).

With PL/SQL, you can combine all these SQL statements into a single program unit. The application can send the entire block to the database instead of sending the SQL statements one at a time. This significantly reduces the number of database calls. As the slide illustrates, if the application is SQL intensive, you can use PL/SQL blocks to group SQL statements before sending them to the Oracle database server for execution.



**Modularized program development**: A basic unit in all PL/SQL programs is the block. Blocks can be in a sequence or they can be nested in other blocks. Modularized program development has the following advantages:

* You can group logically related statements within blocks.
* You can nest blocks inside larger blocks to build powerful programs.
* You can break your application into smaller modules. If you are designing a complex application, PL/SQL allows you to break down the application into smaller, manageable, and logically related modules.

• You can easily maintain and debug the code.

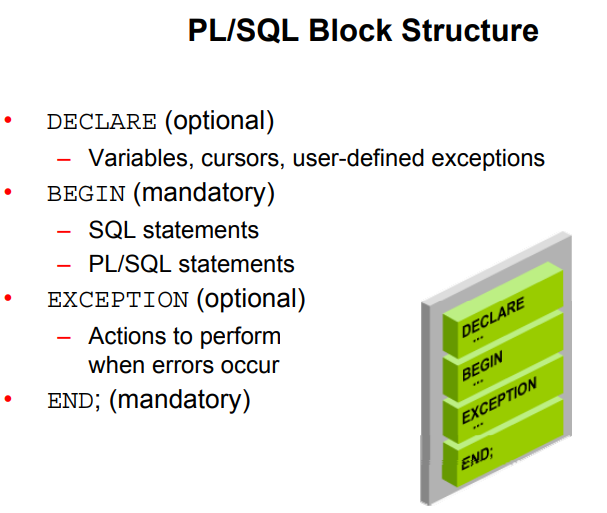
In PL/SQL, modularization is implemented using procedures, functions, and packages, which are discussed in the lesson titled “Creating Stored Procedures and Functions.”

**Integration with tools:** The PL/SQL engine is integrated in Oracle tools such as Oracle Forms, Oracle Reports, and so on. When you use these tools, the locally available PL/SQL engine processes the procedural statements; only the SQL statements are passed to the database.

**Portability**: PL/SQL programs can run anywhere an Oracle server runs, irrespective of the operating system and the platform. You do not need to customize them to each new environment. You can write portable program packages and create libraries that can be reused in different environments.

Exception handling: PL/SQL enables you to handle exceptions efficiently. You can define separate blocks for dealing with exceptions. You learn more about exception handling in the lesson titled “Handling Exceptions.”

PL/SQL shares the same data type system as SQL (with some extensions) and uses the same expression syntax.



**PL/SQL Block Structure**

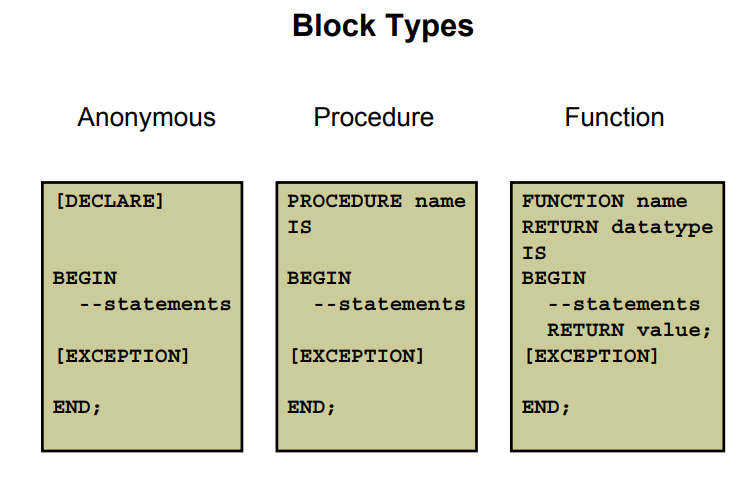
The slide shows a basic PL/SQL block. A PL/SQL block consists of three sections:

• **Declarative (optional):** The declarative section begins with the keyword DECLARE and ends when the executable section starts.

• **Executable (required):** The executable section begins with the keyword BEGIN and ends with END. This section essentially needs to have at least one statement. Observe that END is terminated with a semicolon. The executable section of a PL/SQL block can, in turn, include any number of PL/SQL blocks.

• **Exception handling (optional):** The exception section is nested within the executable section. This section begins with the keyword EXCEPTION.





**Block Types**

A PL/SQL program comprises one or more blocks. These blocks can be entirely separate or nested within another block. There are three types of blocks that make up a PL/SQL program.

They are:

* Anonymous blocks
* Procedures
* Functions

**Anonymous blocks**: Anonymous blocks are unnamed blocks. They are declared inline at the point in an application where they are to be executed and are compiled each time the application is executed. These blocks are not stored in the database. They are passed to the PL/SQL engine for execution at run time. Triggers in Oracle Developer components consist of such blocks.

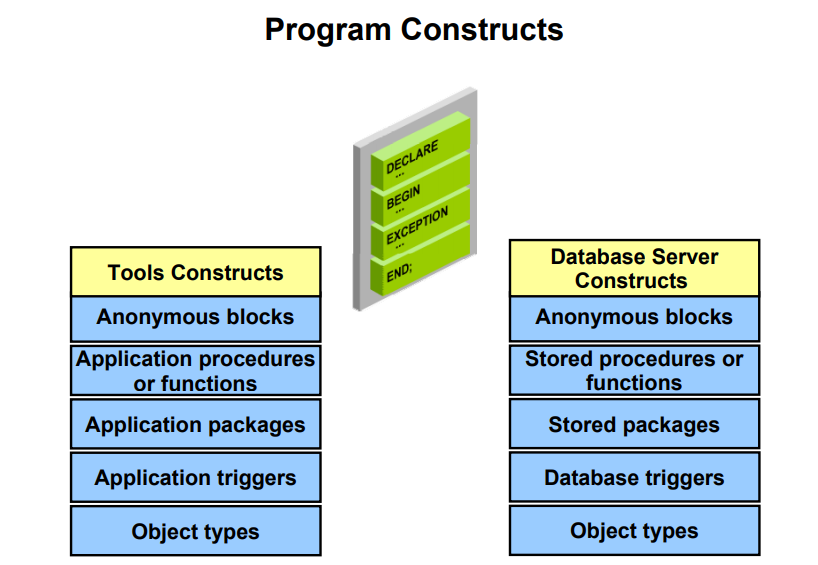
These anonymous blocks get executed at run time because they are inline. If you want to execute the same block again, you have to rewrite the block. You are unable to invoke or call the block that you wrote earlier because blocks are anonymous and do not exist after they are executed.

**Subprograms**: Subprograms are complementary to anonymous blocks. They are named

PL/SQL blocks that are stored in the database. Because they are named and stored, you can invoke them whenever you want (depending on your application). You can declare them either as procedures or as functions. You typically use a procedure to perform an action and a function to compute and return a value.

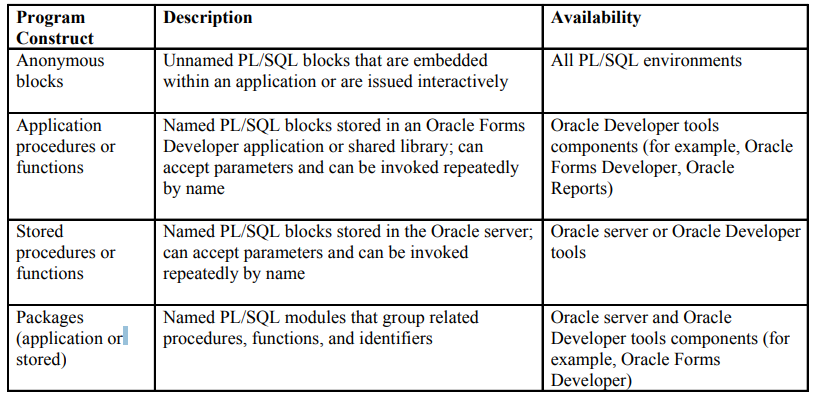
You can store subprograms at the server or application level. Using Oracle Developer components (Forms, Reports), you can declare procedures and functions as part of the application (a form or report) and call them from other procedures, functions, and triggers within the same application whenever necessary.

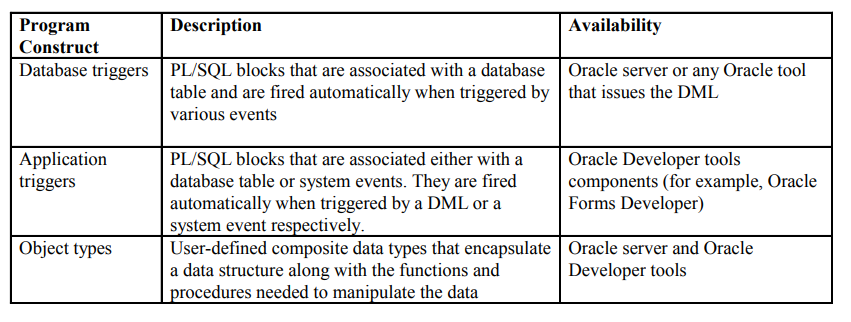
Note: A function is similar to a procedure, except that a function must return a value.



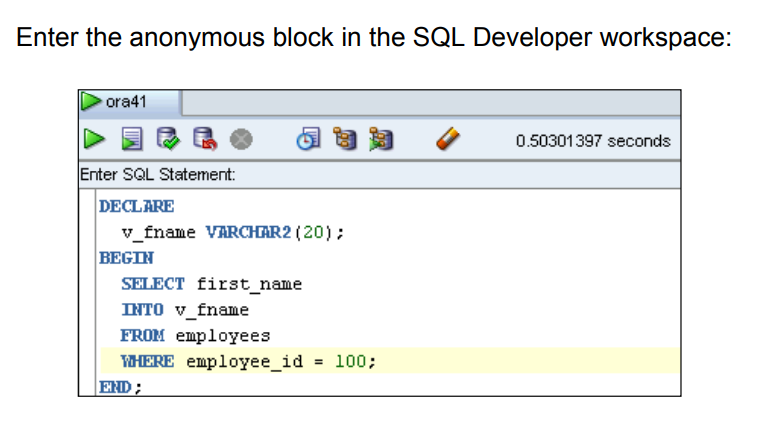
**Program Constructs**

The following table outlines a variety of PL/SQL program constructs that use the basic PL/SQL block. The program constructs are available based on the environment in which they are executed.





**Create an Anonymous Block**

****

DECLARE

V\_fname VARCHAR2(20);

BEGIN

SELECT first\_name

INTO v\_fname

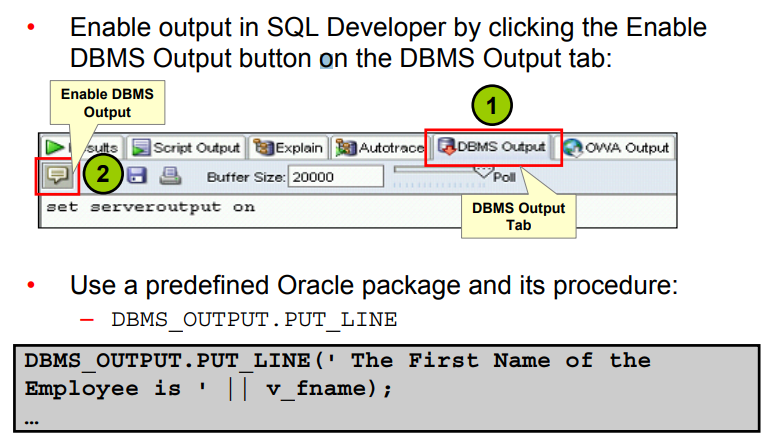
FROM Employees

WHERE employee\_id = 100;

END;

/

**Test the Output of a PL/SQL Block**



PL/SQL does not have built-in input or output functionality. Therefore, you need to use predefined Oracle packages for input and output. To generate output, you must:

• Enable output in SQL Developer by clicking the Enable Output button on the DBMS Output tab. This will, in turn, execute the SET SERVEROUTPUT ON command, which is displayed in the window. To enable output in SQL\*Plus, you must explicitly issue the SET

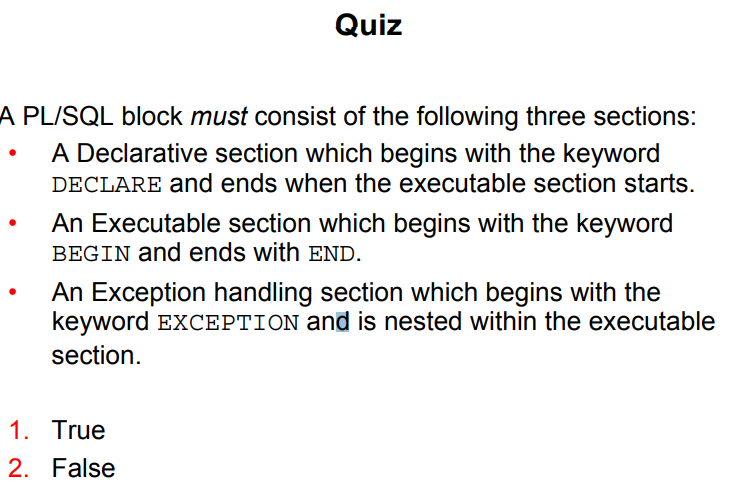
SERVEROUTPUT ON command.

SET SERVEROUTPUT ON;

• Use the PUT\_LINE procedure of the DBMS\_OUTPUT package to display the output. Pass the value that has to be printed as argument to this procedure (as shown in the slide). The procedure then outputs the argument.

DBMS\_OUTPUT.PUT\_LINE(' ‘|| V\_NaME );

**Quiz**



A PL/SQL block consists of three sections:

• Declarative (optional): The optional declarative section begins with the keyword

DECLARE and ends when the executable section starts.

• Executable (required): The required executable section begins with the keyword BEGIN and ends with END. This section essentially needs to have at least one statement. Observe that END is terminated with a semicolon. The executable section of a PL/SQL block can, in turn, include any number of PL/SQL blocks.

• Exception handling (optional): The optional exception section is nested within the executable section. This section begins with the keyword EXCEPTION.

**Practice 1**

The labs folder will be your working directory. You can save your scripts in the labs folder.

Please take the instructor’s help to locate the labs folder for this course. The solutions for all practices are in the soln folder.

1. Which of the following PL/SQL blocks execute successfully?

a. BEGIN

END;

b. DECLARE

amount INTEGER(10);

END;

c. DECLARE

BEGIN

END;

d. DECLARE

amount INTEGER(10);

BEGIN

DBMS\_OUTPUT.PUT\_LINE(amount);

END;

/

Sali >> d.

2. Create and execute a simple anonymous block that outputs “Hello World.” Execute and

save this script as lab\_01\_02\_soln.sql.

Sali >>

SET SERVEROUTPUT ON;

DECLARE

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hello Sali :\* :) :\*' );

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hello Sali :\* :) :\*' );

/ GIVES ERROR – its looking for the END;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hello Sali :\* :) :\*' );

END;

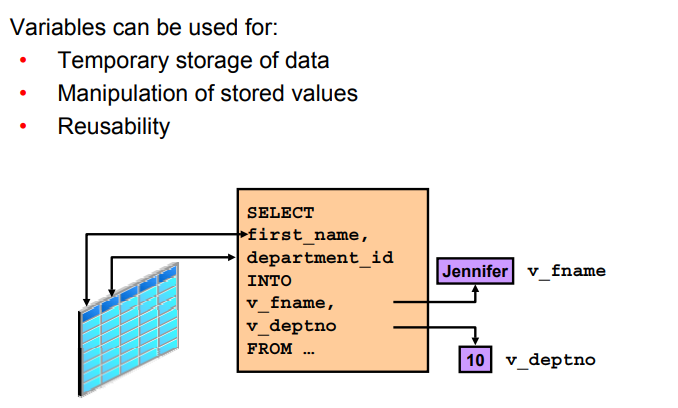
**Variables**

With PL/SQL, you can declare variables and then use them in SQL and procedural statements.

Variables are mainly used for storage of data and manipulation of stored values. Consider the PL/SQL statement shown in the slide. The statement retrieves the first\_name and department\_id from the table. If you have to manipulate the first\_name or the department\_id, then you have to store the retrieved value. Variables are used to temporarily

store the value. You can use the value stored in these variables for processing and manipulating the data. Variables can store any PL/SQL object, such as variables, types, cursors, and subprograms.

**Reusability** is another advantage of declaring variables. After the variables are declared, you can use them repeatedly in an application by referring to them multiple times in various statements.



A variable name:

• Must start with a letter

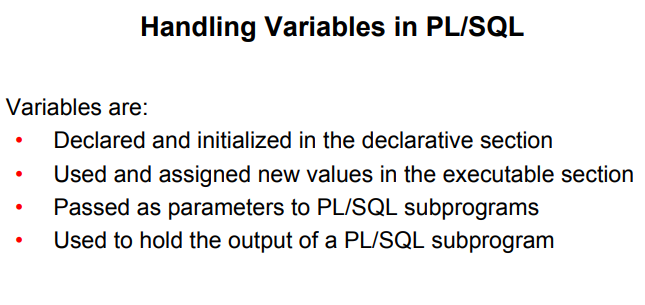
• Can include letters or numbers - no number in the beginning

• Can include special characters (such as $, \_, and # ) - no special characters in the beginning

• Must contain no more than 30 characters

• Must not include reserved words





**Handling Variables in PL/SQL**

You can use variables in the following ways.

• Declare and initialize them in the declaration section: You can declare variables in the declarative part of any PL/SQL block, subprogram, or package. Declarations allocate storage space for a value, specify its data type, and name the storage location so that you can reference it. Declarations can also assign an initial value and impose the NOT NULL constraint on the variable. Forward references are not allowed. You must declare a variable before referencing it in other statements, including other declarative statements.

• Use them and assign new values to them in the executable section: In the executable section, the existing value of the variable can be replaced with the new value.

• Pass them as parameters to PL/SQL subprograms: Subprograms can take parameters.

You can pass variables as parameters to subprograms.

• Use them to hold the output of a PL/SQL subprogram: Variables can be used to hold the value that is returned by a function.

Declaring and Initializing PL/SQL Variables



You must declare all PL/SQL identifiers in the declaration section before referencing them in the PL/SQL block. You have the option of assigning an initial value to a variable (as shown in the slide). You do not need to assign a value to a variable in order to declare it. If you refer to other variables in a declaration, be sure that they are already declared separately in a previous statement.

In the syntax:

**identifier** Is the name of the variable

**CONSTANT**  Constrains the variable so that its value cannot change Constants must be initialized.)

**data type**  Is a scalar, composite, reference, or LOB data type (This course covers only scalar, composite, and LOB data types.)

**NOT NULL** Constrains the variable so that it must contain a value (NOT NULL variables must be initialized.)

**expr** Is any PL/SQL expression that can be a literal expression, another variable, or an expression involving operators and functions

Note: In addition to variables, you can also declare cursors and exceptions in the declarative section. You learn about declaring cursors in the lesson titled “Using Explicit Cursors” and about exceptions in the lesson titled “Handling Exceptions.”

**Declaring and Initializing PL/SQL Variables**

DECLARE

v\_myName VARCHAR2(20);

BEGIN

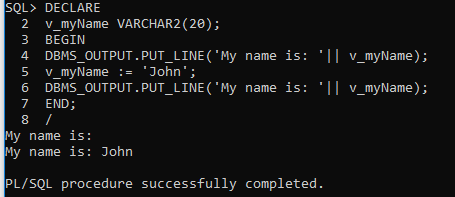
DBMS\_OUTPUT.PUT\_LINE('My name is: '|| v\_myName);

v\_myName := 'John';

DBMS\_OUTPUT.PUT\_LINE('My name is: '|| v\_myName);

END;

/



Sali >> v\_myName did not have value in the first print but after initiation it has “john” inside it

DECLARE

v\_myName VARCHAR2(20):= 'John';

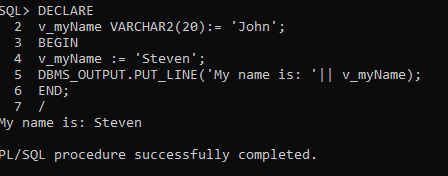
BEGIN

v\_myName := 'Steven';

DBMS\_OUTPUT.PUT\_LINE('My name is: '|| v\_myName);

END;

/



Sali >> v\_myName had “John” inside it which got override by “Steven” inside BEGIN block

DECLARE

v\_event VARCHAR2(15);

BEGIN

v\_event := q'!Father's day!';

DBMS\_OUTPUT.PUT\_LINE('3rd Sunday in June is : '|| v\_event );

v\_event := q'[Mother's day]';

DBMS\_OUTPUT.PUT\_LINE('2nd Sunday in May is : '|| v\_event );

END;

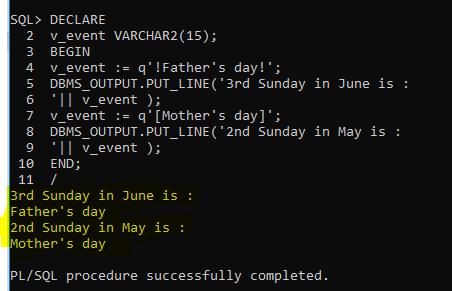
/

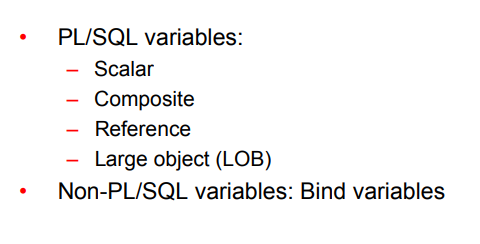
Sali >> 3rd Sunday in June is :

Father’s day!

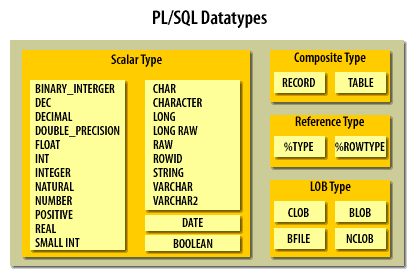
2nd Sunday in May is :

Mother’s day



Types of Variables

Every PL/SQL variables has a data type, which specifies a storage format, constraints, and a valid range of values. PL/SQL supports five data type categories—scalar, composite, reference, large object (LOB), and object—that you can use to declare variables, constants, and pointers.



• **Scalar data types**: Scalar data types hold a single value. The value depends on the data type of the variable. For example, the v\_myName variable in the example in the section “Declaring and Initializing PL/SQL Variables” (in this lesson) is of type VARCHAR2.

Therefore, v\_myName can hold a string value. PL/SQL also supports Boolean variables.

These data types don’t include any internal components. It includes data types such as NUMBER, DATE, BOOLEAN, etc.

• **Composite data types**: Composite data types contain internal elements that are either scalar or composite. RECORD, collections and TABLE are examples of composite data types.

• **Reference data types**: Reference data types hold values, called pointers that point to a storage location.

• **LOB data types**: Large Objects data types hold values, called locators that specify the location of large objects (such as graphic images) that are stored outside the table.

This type of data type stores objects that are relatively large in size and stored separately from other data types such as text, graphic images, video clips, sound, etc.

Now, I have listed down some of the sub data types frequently used in PL/SQL

1. **Numeric**: Numeric values on which arithmetic operations are performed. It includes sub types such as number, decimal, real, float, etc.

2. **Character**: Character values on which character operations such as strings are performed. It includes sub types such as char, varchar, varchar2, nvarchar2, etc.

3. **Data and Time**: This data type is used to store fixed data type which displays and saves time and date values. The default data format saved into the database might be ‘DD-MM-YY’. However, you can change and alter the position of the terms accordingly.

4. **Boolean**: These include logical values on which logical operations are performed. The logical values are the Boolean values TRUE and FALSE and the value NULL. But, SQL has no data type equivalent to BOOLEAN. It cannot be used in SQL Statements, built in SQL functions such as To\_char, PL/SQL function invoked from DQL commands.

5. **Number**: Syntax: Number(Precision, Scale). Fixed-point or floating-point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0.

6. **Float**: ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits).

7. **Integer**: ANSI and IBM specific integer type with maximum precision of 38 decimal digits

8. **Real**: Floating-point type with maximum precision of 63 binary digits (approximately 18 decimal digits).

9. **Varchar2**: Variable-length character string with maximum size of 32,767 bytes.

10. **Rowid**: Physical row identifier, the address of a row in an ordinary table.

11. **Bfile**: Used to store large binary objects in operating system files outside the database System-dependent. Cannot exceed 4GB.

12. **Blob**: Used to store large binary objects in the database. Memory Capacity: 8 to 128 TB.

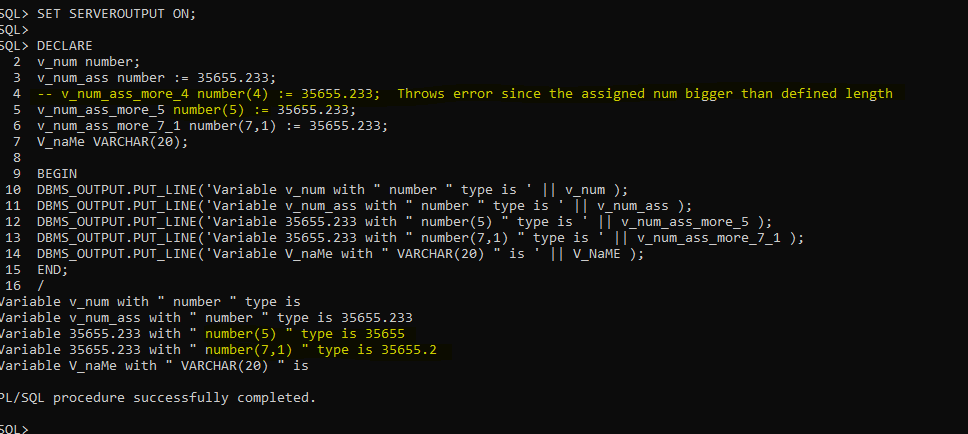
13. **Clob**: Used to store large blocks of character data in the database. Memory Capacity: 8 to 128 TB.

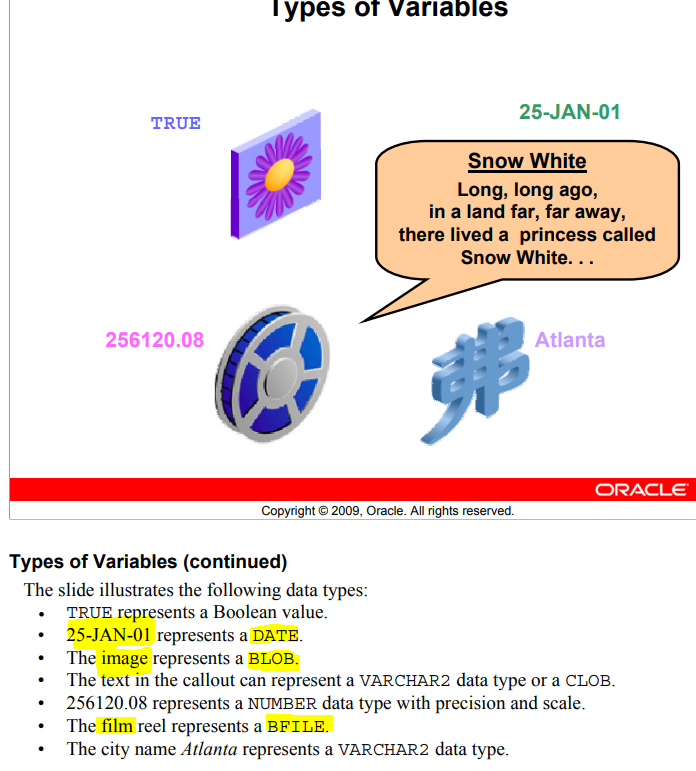
14. **Nclob**: Used to store large blocks of NCHAR data in the database. Memory Capacity: 8 to 128 TB.

Note: A variable in PL/SQL program code can be assigned a NULL value i.e., ‘’. However, it cannot be equated with any other variable or even itself.

Non-PL/SQL variables include host language variables declared in precompiler programs, screen fields in Forms applications, and host variables. You learn about host variables later in this lesson.

For more information about LOBs, see the PL/SQL User’s Guide and Reference.

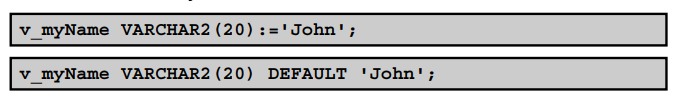




Initialize variables designated as NOT NULL and CONSTANT.

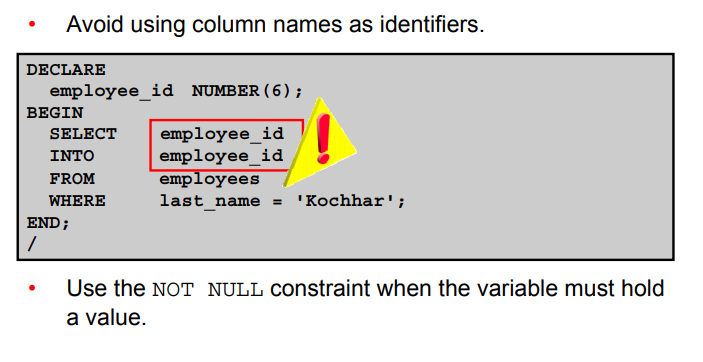
• Initialize variables with the assignment operator (:=) or the

DEFAULT keyword:



• Declare one identifier per line for better readability and code maintenance.

sal CONSTANT NUMBER := 50000.00;



**Guidelines for Declaring PL/SQL Variables**

• Initialize the variable to an expression with the assignment operator (:=) or with the DEFAULT reserved word. If you do not assign an initial value, the new variable contains NULL by default until you assign a value. To assign or reassign a value to a variable, you write a PL/SQL assignment statement. It is good programming practice to initialize all

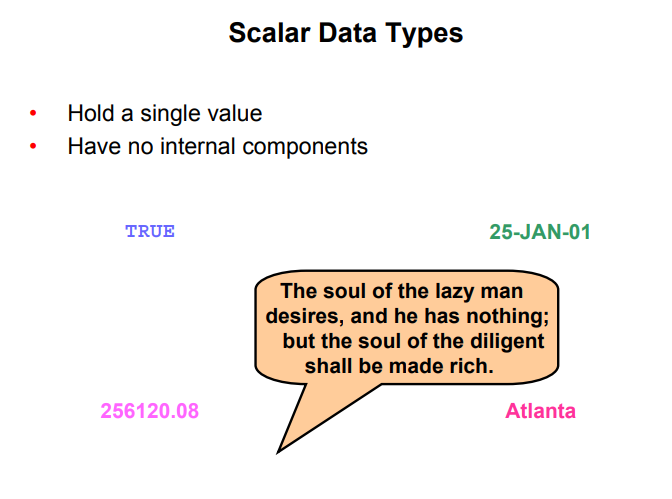
variables.

• Two objects can have the same name only if they are defined in different blocks. Where they coexist, you can qualify them with labels and use them.

• Avoid using column names as identifiers. If PL/SQL variables occur in SQL statements and have the same name as a column, the Oracle server assumes that it is the column that is being referenced. Although the code example in the slide works, code that is written using the same name for a database table and a variable is not easy to read or maintain.

• Impose the NOT NULL constraint when the variable must contain a value. You cannot assign nulls to a variable defined as NOT NULL. The NOT NULL constraint must be followed by an initialization clause.

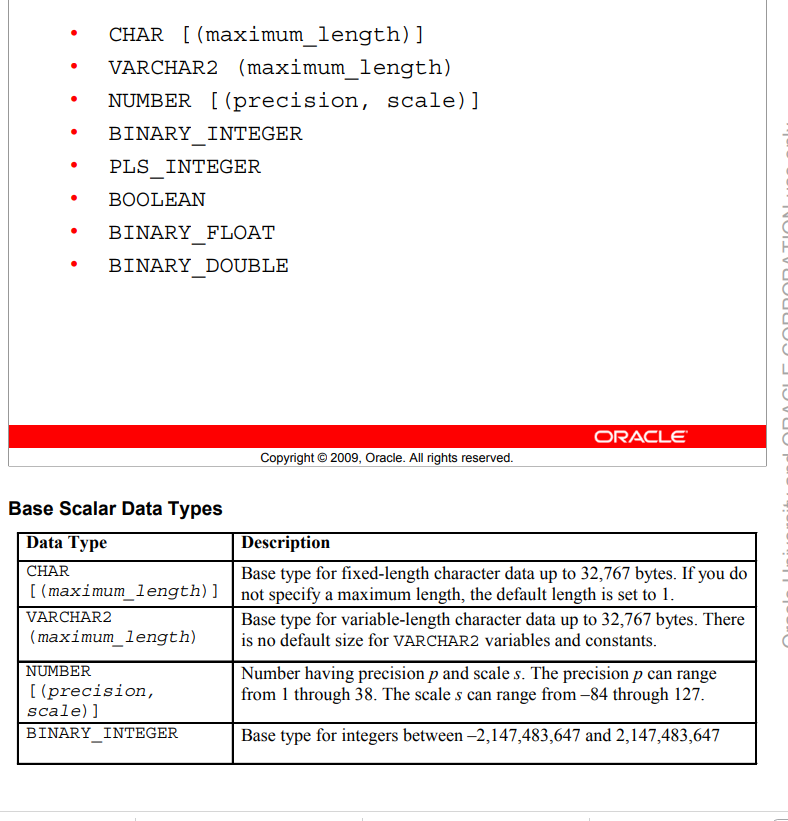
pincode VARCHAR2(15) NOT NULL := 'Oxford';

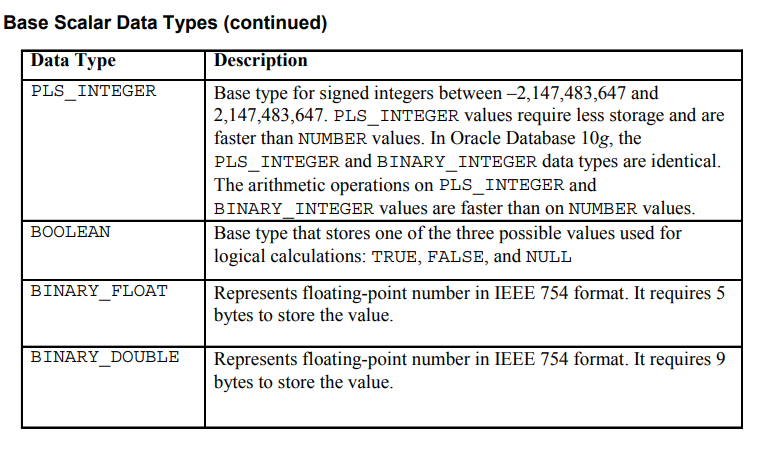


**Scalar Data Types**

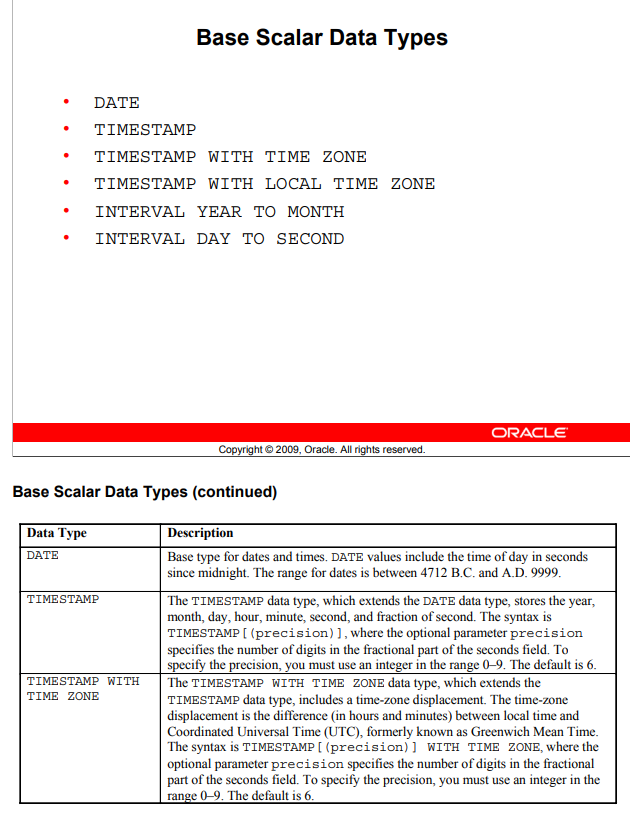
PL/SQL provides a variety of predefined data types. For instance, you can choose from integer, floating point, character, Boolean, date, collection, and LOB types. This lesson covers the basic types that are used frequently in PL/SQL programs.

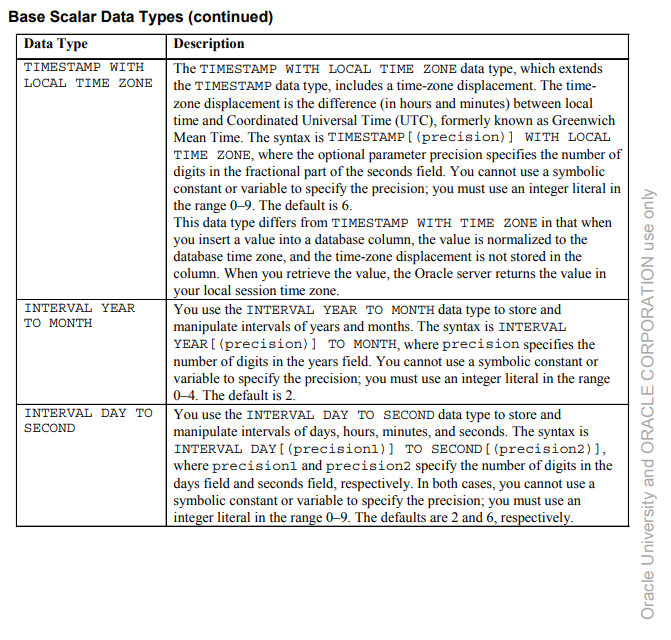
A **scalar data** type holds a single value and has no internal components. Scalar data types can be classified into four categories: **number**, **character**, **date**, and **Boolean**. Character and number data types have subtypes that associate a base type to a constraint. For example, NTEGER and POSITIVE are subtypes of the NUMBER base type.



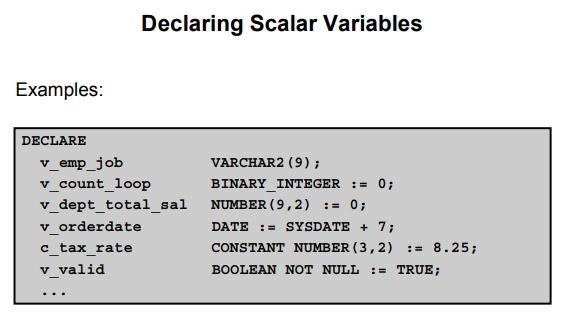


Base Scalar Data Types





Declaring Scalar Variables



The examples of variable declaration shown in the slide are defined as follows.

• **v\_emp\_job**: Variable to store an employee job title

• **v\_count\_loop**: Variable to count the iterations of a loop; initialized to 0

• **v\_dept\_total\_sal**: Variableto accumulatethe total salary for a department; initialized to 0

• **v\_orderdate**: Variable to store the ship date of an order; initialized to one week from today

• **c\_tax\_rate**: Constant variable for the tax rate (which never changes throughout the PL/SQL block); set to 8.25

• **v\_valid**: Flag to indicate whether a piece of data is valid or invalid; initialized to TRUE