

11

Including Constraints

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	45 minutes	Lecture
	25 minutes	Practice
	70 minutes	Total

Objectives

At the end of this lesson, you will be able to:

- **Describe constraints**
- **Create and maintain constraints**

Lesson Aim

In this lesson, you will learn how to implement business rules by including integrity constraints.

What Are Constraints?

- **Constraints enforce rules at the table level.**
- **Constraints prevent the deletion of a table if there are dependencies.**
- **The following constraint types are valid in Oracle8:**
 - **NOT NULL**
 - **UNIQUE Key**
 - **PRIMARY KEY**
 - **FOREIGN KEY**
 - **CHECK**

11-3

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Constraints

Oracle8 Server uses *constraints* to prevent invalid data entry into tables.

You can use constraints to do the following:

- Enforce rules at the table level whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables.
- Provide rules for Oracle tools, such as Developer/2000.

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that this column may not contain a null value
UNIQUE Key	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	Specifies a condition that must be true

For more information, see *Oracle8 Server SQL Reference, Release 8.0*, "CONSTRAINT Clause."



Constraint Guidelines

- **Name a constraint or the Oracle8 Server will generate a name by using the SYS_Cn format.**
- **Create a constraint:**
 - **At the same time as the table is created**
 - **After the table has been created**
- **Define a constraint at the column or table level.**
- **View a constraint in the data dictionary.**

Constraint Guidelines

All constraints are stored in the data dictionary. Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object naming rules. If you do not name your constraint, Oracle8 generates a name with the format SYS_Cn, where *n* is an integer to create a unique constraint name.

Constraints can be defined at the time of table creation or after the table has been created.

You can view the constraints defined for a specific table by looking at the USER_CONSTRAINTS data dictionary table.

Defining Constraints

```
CREATE TABLE [schema.]table
    (column datatype [DEFAULT expr]
    [column_constraint],
    ...
    [table_constraint]);
```

```
CREATE TABLE emp(
    (empno    NUMBER(4),
    ename     VARCHAR2(10),
    ...
    deptno    NUMBER(7,2) NOT NULL,
    CONSTRAINT emp_empno_pk
                PRIMARY KEY (EMPNO));
```

11-5

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Defining Constraints

The slide gives the syntax for defining constraints while creating a table.

In the syntax:

<i>schema</i>	is the same as the owner's name
<i>table</i>	is the name of the table
DEFAULT <i>expr</i>	specifies a default value if a value is omitted in the INSERT statement
<i>column</i>	is the name of the column
<i>datatype</i>	is the column's datatype and length
<i>column_constraint</i>	is an integrity constraint as part of the column definition
<i>table_constraint</i>	is an integrity constraint as part of the table definition

For more information, see

Oracle8 Server SQL Reference, Release 8.0, "CREATE TABLE."



Defining Constraints

- Column constraint level

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table constraint level

```
column, ...  
  [CONSTRAINT constraint_name] constraint_type  
  (column, ...),
```

Defining Constraints (continued)

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also temporarily disabled.

Constraints can be defined at one of two levels.

Constraint Level	Description
Column	References a single column and is defined within a specification for the owning column; can define any type of integrity constraint
Table In the syntax: <i>constraint_name</i> <i>constraint_type</i>	References one or more columns and is defined separately from the definitions of the columns in the table; can define any constraints except NOT NULL is the name of the constraint is the type of the constraint

Class Management Note


Explain that the column level and the table level refer to location in the syntax.


The NOT NULL Constraint

Ensures that null values are not permitted for the column

EMP

EMPNO	ENAME	JOB	...	COMM	DEPTNO
7839	KING	PRESIDENT			10
7698	BLAKE	MANAGER			30
7782	CLARK	MANAGER			10
7566	JONES	MANAGER			20
...					


NOT NULL constraint
(no row may contain
a null value for
this column)


**Absence of NOT NULL
constraint**
(any row can contain
null for this column)


NOT NULL constraint

The NOT NULL Constraint

The NOT NULL constraint ensures that null values are not allowed in the column. Columns without the NOT NULL constraint can contain null values by default.

The NOT NULL Constraint

Defined at the column level

```
SQL> CREATE TABLE emp(  
 2      empno      NUMBER(4),  
 3      ename      VARCHAR2(10) NOT NULL,  
 4      job        VARCHAR2(9),  
 5      mgr        NUMBER(4),  
 6      hiredate   DATE,  
 7      sal        NUMBER(7,2),  
 8      comm       NUMBER(7,2),  
 9      deptno     NUMBER(7,2) NOT NULL);
```

11-8

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The NOT NULL Constraint (continued)

The NOT NULL constraint can be specified only at the column level, not at the table level.

The example above applies the NOT NULL constraint to the ENAME and DEPTNO columns of the EMP table. Because these constraints are unnamed, Oracle8 Server will create names for them.

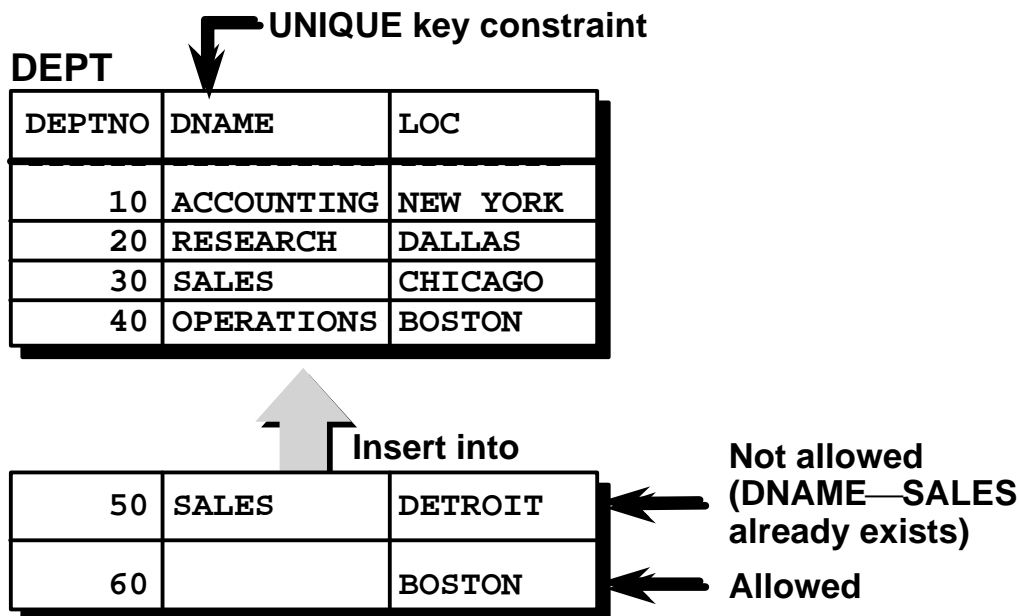
You can specify the name of the constraint while specifying the constraint.

```
... deptno NUMBER(7,2)
```

```
CONSTRAINT emp_deptno_nn NOT NULL;
```

Note: All the constraint examples described in this lesson may not be present in the sample tables provided with the course. If desired, these constraints can be added to the tables.

The UNIQUE Key Constraint



11-9

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The UNIQUE Key Constraint

A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique—that is, no two rows of a table have duplicate values in a specified column or set of columns. The column (or set of columns) included in the definition of the UNIQUE key constraint is called the *unique key*. If the UNIQUE key comprises more than one column, that group of columns is said to be a *composite unique key*.

UNIQUE key constraints allow the input of nulls unless you also define NOT NULL constraints for the same columns. In fact, any number of rows can include nulls for columns without NOT NULL constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite UNIQUE key) always satisfies a UNIQUE key constraint.

Note: Because of the search mechanism for UNIQUE constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite UNIQUE key constraint.

Class Management Note

Explain to students that since SALES department already exists the first entry is not allowed. In the second entry, the department name is null; this entry is allowed.

The UNIQUE Key Constraint

Defined at either the table level or the column level

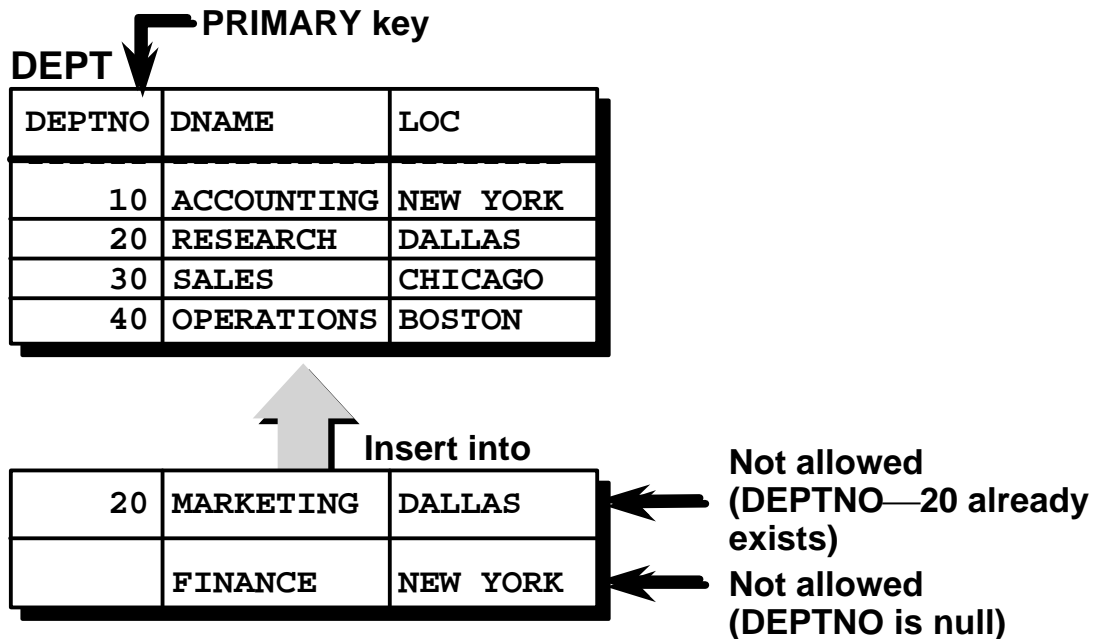
```
SQL> CREATE TABLE dept(  
2      deptno    NUMBER(2),  
3      dname     VARCHAR2(14),  
4      loc       VARCHAR2(13),  
5      CONSTRAINT dept_dname_uk UNIQUE);
```

The UNIQUE Key Constraint (continued)

UNIQUE key constraints can be defined at the column or table level. A composite unique key is created by using the table level definition.

The example on the slide applies UNIQUE key constraint to the DNAME column of the DEPT table. The name of the constraint is DEPT_DNAME_UK.

The PRIMARY KEY Constraint



11-11

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The PRIMARY KEY Constraint

A PRIMARY KEY constraint creates a primary key for the table. Only one primary key can be created for a each table. The PRIMARY KEY constraint is a column or set of columns that uniquely identifies each row in a table. This constraint enforces uniqueness of the column or column combination and ensures that no column that is part of the primary key can contain a null value.

The PRIMARY KEY Constraint

Defined at either the table level or the column level

```
SQL> CREATE TABLE dept(  
2     deptno    NUMBER(2),  
3     dname     VARCHAR2(14),  
4     loc       VARCHAR2(13),  
5     CONSTRAINT dept_dname_uk UNIQUE,  
6     CONSTRAINT dept_deptno_pk PRIMARY KEY);
```

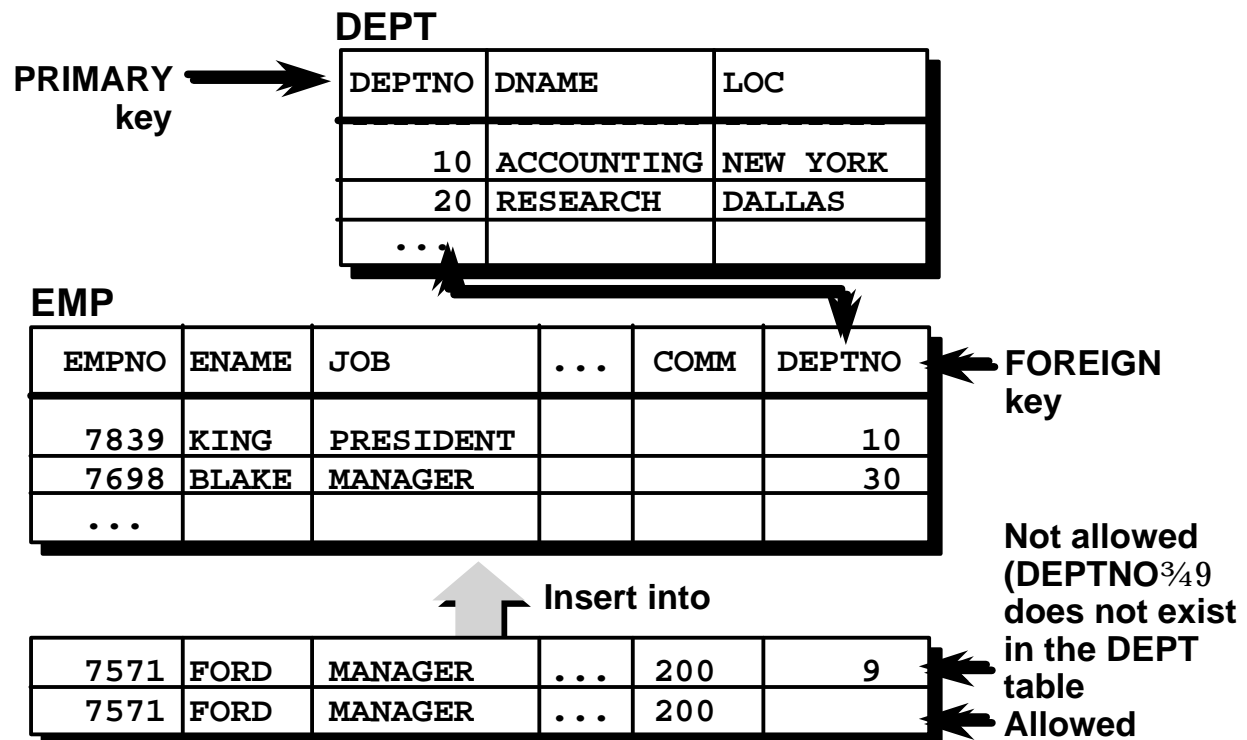
The PRIMARY KEY Constraint (continued)

PRIMARY KEY constraints can be defined at the column level or table level. A composite PRIMARY KEY is created by using the table level definition.

The example on the slide defines a PRIMARY KEY constraint on the DEPTNO column of the DEPT table. The name of the constraint is DEPT_DEPTNO_PK.

Note: A UNIQUE index is automatically created for a PRIMARY KEY column.

The FOREIGN KEY Constraint



11-13

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The FOREIGN KEY Constraint

The FOREIGN KEY, or referential integrity constraint, designates a column or combination of columns as a foreign key and establishes a relationship between a primary key or a unique key in the same table or a different table. In the example on the slide, DEPTNO has been defined as the foreign key in the EMP table (dependent or child table); it references the DEPTNO column of the DEPT table (referenced or parent table).

A foreign key value must match an existing value in the parent table or be NULL.

Foreign keys are based on data values and are purely logical, not physical, pointers.



A foreign key that is part of a primary key cannot be a null value because no part of a primary key can be NULL.



The FOREIGN KEY Constraint

Defined at either the table level or the column level

```
SQL> CREATE TABLE emp(  
 2      empno      NUMBER(4),  
 3      ename      VARCHAR2(10) NOT NULL,  
 4      job        VARCHAR2(9),  
 5      mgr        NUMBER(4),  
 6      hiredate   DATE,  
 7      sal        NUMBER(7,2),  
 8      comm       NUMBER(7,2),  
 9      deptno     NUMBER(7,2) NOT NULL,  
10      CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)  
11                  REFERENCES dept (deptno));
```

11-14

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The FOREIGN KEY Constraint (continued)

FOREIGN KEY constraints can be defined at the column or table constraint level. A composite foreign key is created by using the table-level definition.

The example on the slide defines a FOREIGN KEY constraint on the DEPTNO column of the EMP table. The name of the constraint is EMP_DEPTNO_FK.

FOREIGN KEY Constraint

Keywords

- **FOREIGN KEY**
 - Defines the column in the child table at the table constraint level
- **REFERENCES**
 - Identifies the table and column in the parent table
- **ON DELETE CASCADE**
 - Allows deletion in the parent table and deletion of the dependent rows in the child table

11-15

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The FOREIGN KEY Constraint (continued)

The foreign key is defined in the child table, and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- FOREIGN KEY is used to define the column in the child table at the table constraint level.
- REFERENCES identifies the table and column in the parent table.
- ON DELETE CASCADE indicates that when the row in the parent table is deleted, the dependent rows in the child table will also be deleted.

Without the ON DELETE CASCADE option, the row in the parent table cannot be deleted if it is referenced in the child table.

The CHECK Constraint

- Defines a condition that each row must satisfy
- Expressions that are not allowed:
 - References to pseudocolumns CURRVAL, NEXTVAL, LEVEL, and ROWNUM
 - Calls to SYSDATE, UID, USER, and USERENV functions
 - Queries that refer to other values in other rows

```
..., deptno  NUMBER(2),  
        CONSTRAINT emp_deptno_ck  
        CHECK (DEPTNO BETWEEN 10 AND 99),...
```

11-16

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The CHECK Constraint

The CHECK constraint defines a condition that each row must satisfy. The condition can use the same constructs as query conditions, with the following exceptions:

- References to the CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
- Calls to SYSDATE, UID, USER, and USERENV functions
- Queries that refer to other values in other rows

A single column can have multiple CHECK constraints that reference the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column.

CHECK constraints can be defined at the column level or table level.

Class Management Note

The DISABLE option in the definition of an integrity constraint means that the Oracle8 Server does not enforce the constraint and simply documents it. Enable the constraint by using the ENABLE clause.

Adding a Constraint

```
ALTER TABLE table  
ADD [CONSTRAINT constraint] type (column);
```

- Add or drop, but not modify, a constraint
- Enable or disable constraints
- Add a NOT NULL constraint by using the MODIFY clause

11-17

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Adding a Constraint

You can add a constraint for existing tables by using the ALTER TABLE statement with the ADD clause.

In the syntax:

<i>table</i>	is the name of the table
<i>constraint</i>	is the name of the constraint
<i>type</i>	is the constraint type
<i>column</i>	is the name of the column affected by the constraint

The constraint name syntax is optional, although recommended. If you do not name your constraints, the system will generate constraint names.



Guidelines

- You can add, drop, enable, or disable a constraint, but you cannot modify its structure.
- You can add a NOT NULL constraint to an existing column by using the MODIFY clause of the ALTER TABLE statement.

Adding a Constraint

Add a FOREIGN KEY constraint to the EMP table indicating that a manager must already exist as a valid employee in the EMP table.

```
SQL> ALTER TABLE      emp
      2  ADD CONSTRAINT emp_mgr_fk
      3          FOREIGN KEY(mgr) REFERENCES emp(empno);
Table altered.
```

Adding a Constraint

The example above creates a FOREIGN KEY constraint on the EMP table. The constraint ensures that a manager exists as a valid employee in the EMP table.

Dropping a Constraint

- Remove the manager constraint from the EMP table.

```
SQL> ALTER TABLE emp
2 DROP CONSTRAINT emp_mgr_fk;
Table altered.
```

- Remove the PRIMARY KEY constraint on the DEPT table and drop the associated FOREIGN KEY constraint on the EMP.DEPTNO column.

```
SQL> ALTER TABLE dept
2 DROP PRIMARY KEY CASCADE;
Table altered.
```

11-19

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Dropping a Constraint

To drop a constraint, you can identify the constraint name from the USER_CONSTRAINTS and USER_CONS_COLUMNS data dictionary views. Then use the ALTER TABLE statement with the DROP clause. The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

Syntax

```
ALTER TABLE table
DROP PRIMARY KEY | UNIQUE (column) |
CONSTRAINT constraint [CASCADE];
```

where:	<i>table</i>	is the name of the table
	<i>column</i>	is the name of the column affected by the constraint
	<i>constraint</i>	is the name of the constraint

When you drop an integrity constraint, that constraint is no longer enforced by the Oracle8 Server and is no longer available in the data dictionary.



Disabling Constraints

- Execute the **DISABLE** clause of the **ALTER TABLE** statement to deactivate an integrity constraint.
- Apply the **CASCADE** option to disable dependent integrity constraints.

```
SQL> ALTER TABLE          emp
      2  DISABLE CONSTRAINT  emp_empno_pk CASCADE;
Table altered.
```

Disabling a Constraint

You can disable a constraint without dropping it or recreating it by using the **ALTER TABLE** statement with the **DISABLE** clause.

Syntax

```
ALTER TABLE table
DISABLE CONSTRAINT constraint [CASCADE];
```

where: *table* is the name of the table
constraint is the name of the constraint

Guidelines

- You can use the **DISABLE** clause in both the **CREATE TABLE** statement and the **ALTER TABLE** statement.
- The **CASCADE** clause disables dependent integrity constraints.

Enabling Constraints

- **Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.**

```
SQL> ALTER TABLE          emp
      2  ENABLE CONSTRAINT    emp_empno_pk;
Table altered.
```

- **A UNIQUE or PRIMARY KEY index is automatically created if you enable a UNIQUE or PRIMARY KEY constraint.**

Enabling a Constraint

You can enable a constraint without dropping it or recreating it by using the ALTER TABLE statement with the ENABLE clause.

Syntax

```
ALTER TABLE table
ENABLE CONSTRAINT constraint;
```

where: *table* is the name of the table
constraint is the name of the constraint

Guidelines

- If you enable a constraint, that constraint applies to all the data in the table. All the data in the table must fit the constraint.
- If you enable a UNIQUE or PRIMARY KEY constraint, a UNIQUE or PRIMARY KEY index is automatically created.
- You can use the ENABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.

Viewing Constraints

Query the USER_CONSTRAINTS table to view all constraint definitions and names.

```
SQL>  SELECT  constraint_name, constraint_type,  
2          search_condition  
3  FROM    user_constraints  
4  WHERE   table_name = 'EMP';
```

CONSTRAINT_NAME	C SEARCH_CONDITION
-----	-----
SYS_C00674	C EMPNO IS NOT NULL
SYS_C00675	C DEPTNO IS NOT NULL
EMP_EMPNO_PK	P
...	

Viewing Constraints

After creating a table, you can confirm its existence by issuing a DESCRIBE command. The only constraint that you can verify is the NOT NULL constraint. To view all constraints on your table, query the USER_CONSTRAINTS table.

The example above displays all the constraints on the EMP table.

Note: Constraints that are not named by the table owner receive the system-assigned constraint name. In constraint type, C stands for CHECK, P stands for PRIMARY KEY, R for referential integrity, and U for UNIQUE KEY. Notice that the NULL constraint is really a CHECK constraint.

Viewing the Columns Associated with Constraints

View the columns associated with the
constraint names in the
USER_CONS_COLUMNS view

```
SQL> SELECT  constraint_name, column_name
2  FROM      user_cons_columns
3  WHERE     table_name = 'EMP';
```

CONSTRAINT_NAME	COLUMN_NAME
-----	-----
EMP_DEPTNO_FK	DEPTNO
EMP_EMPNO_PK	EMPNO
EMP_MGR_FK	MGR
SYS_C00674	EMPNO
SYS_C00675	DEPTNO

Viewing Constraints

You can view the names of the columns involved in constraints by querying the USER_CONS_COLUMNS data dictionary view. This view is especially useful for constraints that use the system-assigned name.

Summary

- **Create the following types of constraints:**
 - **NOT NULL**
 - **UNIQUE Key**
 - **PRIMARY KEY**
 - **FOREIGN KEY**
 - **CHECK**
- **Query the USER_CONSTRAINTS table to view all constraint definitions and names.**

Summary

Oracle8 Server uses constraints to prevent invalid data entry into tables.

The following constraint types are valid in Oracle8

- NOT NULL
- UNIQUE Key
- PRIMARY KEY
- FOREIGN KEY
- CHECK

You can query the USER_CONSTRAINTS table to view all constraint definitions and names.

Practice Overview

- **Adding constraints to existing tables**
- **Adding additional columns to a table**
- **Displaying information in data dictionary views**

11-25

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Practice Overview

In this practice, you will add constraints and additional columns to a table using the statements covered in this lesson.

Practice 11

1. Add a table level PRIMARY KEY constraint to the EMPLOYEE table using the ID column. The constraint should be enabled at creation.
2. Create a PRIMARY KEY constraint on the DEPARTMENT table using the ID column. The constraint should be enabled at creation.
3. Add a foreign key reference on the EMPLOYEE table that will ensure that the employee is not assigned to a nonexistent department.
4. Confirm that the constraints were added by querying USER_CONSTRAINTS. Note the types and names of the constraints. Save your statement text in a file called *p11q4.sql*.

CONSTRAINT_NAME	C
-----------------	---

DEPARTMENT_ID_PK	P
EMPLOYEE_ID_PK	P
EMPLOYEE_DEPT_ID_FK	R

5. Display the object names and types from the USER_OBJECTS data dictionary view EMPLOYEE and DEPARTMENT tables. You may want to format the columns for readability. Notice that the new tables and a new index were created.

OBJECT_NAME	OBJECT_TYPE
-------------	-------------

DEPARTMENT	TABLE
DEPARTMENT_ID_PK	INDEX
EMPLOYEE	TABLE
EMPLOYEE_ID_PK	INDEX

If you have time, complete the following exercises.

6. Modify the EMPLOYEE table. Add a SALARY column of NUMBER data type, precision 7.

11

Creating Views

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	20 minutes	Lecture
	20 minutes	Practice
	40 minutes	Total

Objectives

At the end of this lesson, you will be able to:

- **Describe a view**
- **Create a view**
- **Retrieve data through a view**
- **Alter the definition of a view**
- **Insert, update, and delete data through a view**
- **Drop a view**

11-28

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Lesson Aim

In this lesson, you will learn to create and use views. You will also learn to query the relevant data dictionary object to retrieve information about views.

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence	Generates primary key values
Index	Improves the performance of some queries
Synonym	Alternative name for an object

What Is a View?

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7839	KING	PRESIDENT				
7782	CLARK	MANAGER				
7934	MILLER	CLERK				
7876	ADAMS	CLERK	7788	12-JAN-83	1100	
7369	SMITH	CLERK	7902	17-DEC-80	800	
7902	FORD	ANALYST	7566	03-DEC-81	3000	
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	

11-30

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

What Is a View?

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called *base tables*. The view is stored as a SELECT statement in the data dictionary.

Why Use Views?

- **To restrict database access**
- **To make complex queries easy**
- **To allow data independence**
- **To present different views of the same data**

11-31

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Advantages of Views

- Restrict access to the database because the view can display a selective portion of the database.
- Allow users to make simple queries to retrieve the results from complicated queries. For example, views allow users to query information from multiple tables without knowing how to write a join statement.
- Provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Provide groups of users access to data according to their particular criteria.

For more information, see

Oracle8 Server SQL Reference, Release 8.0, "CREATE VIEW."



Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML through view	Yes	Not always

Simple Views Versus Complex Views

There are two classifications for views: simple and complex. The basic difference is related to the DML (insert, update, and delete) operations.

- A *simple view* is one that:
 - Derives data from only one table
 - Contains no functions or groups of data
 - Can perform DML through the view
- A *complex view* is the one that:
 - Derives data from many tables
 - Contains functions or groups of data
 - Cannot always perform DML through the view

Creating a View

- You can embed a subquery within the **CREATE VIEW** statement.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY]
```

syntax.

- The subquery cannot contain an **ORDER BY** clause.

Creating a View

You can create a view by embedding a subquery within the CREATE VIEW statement.

In the syntax:

OR REPLACE	recreates the view if it already exists.	
FORCE	creates the view regardless of whether or not the base tables exist.	
NOFORCE	creates the view only if the base tables exist. This is the default.	
<i>view</i>	is the name of the view.	
<i>alias</i>	specifies names for the expressions selected by the view's query. The number of aliases must match the number of expressions selected by	the view.
<i>subquery</i>	is a complete SELECT statement. You can use aliases for the columns in the SELECT list.	
WITH CHECK OPTION	specifies that only rows accessible to the view can be inserted or updated.	
<i>constraint</i>	is the name assigned to the CHECK OPTION constraint.	
WITH READ ONLY	ensures that no DML operations can be performed on this view.	

Creating a View

- Create a view, EMPVU10, that contains details of employees in department 10.

```
SQL> CREATE VIEW      empvu10
  2  AS SELECT        empno, ename, job
  4  FROM              emp
  5  WHERE             deptno = 10;
View created.
```

- Describe the structure of the view by using the SQL*Plus DESCRIBE command.

```
SQL> DESCRIBE empvu10
```

11-34

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Creating a View

The example above creates a view that contains the employee number, name, and job title for all the employees in department 10.

You can display the structure of the view by using the SQL*Plus DESCRIBE command.

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)

Guidelines for Creating a View

- The subquery that defines a view can contain complex SELECT syntax, including joins, groups, and subqueries.
- The subquery that defines the view cannot contain an ORDER BY clause. The ORDER BY clause is specified when you retrieve data from the view.
- If you do not specify a constraint name for a view created with the CHECK OPTION, the system will assign a default name in the format SYS_Cn.
- You can use the OR REPLACE option to change the definition of the view without dropping and recreating it or regranting object privileges previously granted on it.

Creating a View

- **Create a view by using column aliases in the subquery.**

```
SQL> CREATE VIEW      salvu30
  2  AS SELECT        empno EMPLOYEE_NUMBER, ename NAME,
  3                  sal SALARY
  4  FROM              emp
  5  WHERE              deptno = 30;
View created.
```

- **Select the columns from this view by the given alias name.**

Creating a View

You can control the column names by including column aliases within the subquery.

The example above creates a view containing the employee number with the alias EMPLOYEE_NUMBER, name with the alias NAME, and salary with the alias SALARY for department 30.

Alternatively, you can control the column names by including column aliases in the CREATE VIEW clause.

Retrieving Data from a View

```
SQL> SELECT *  
2 FROM salvu30;
```

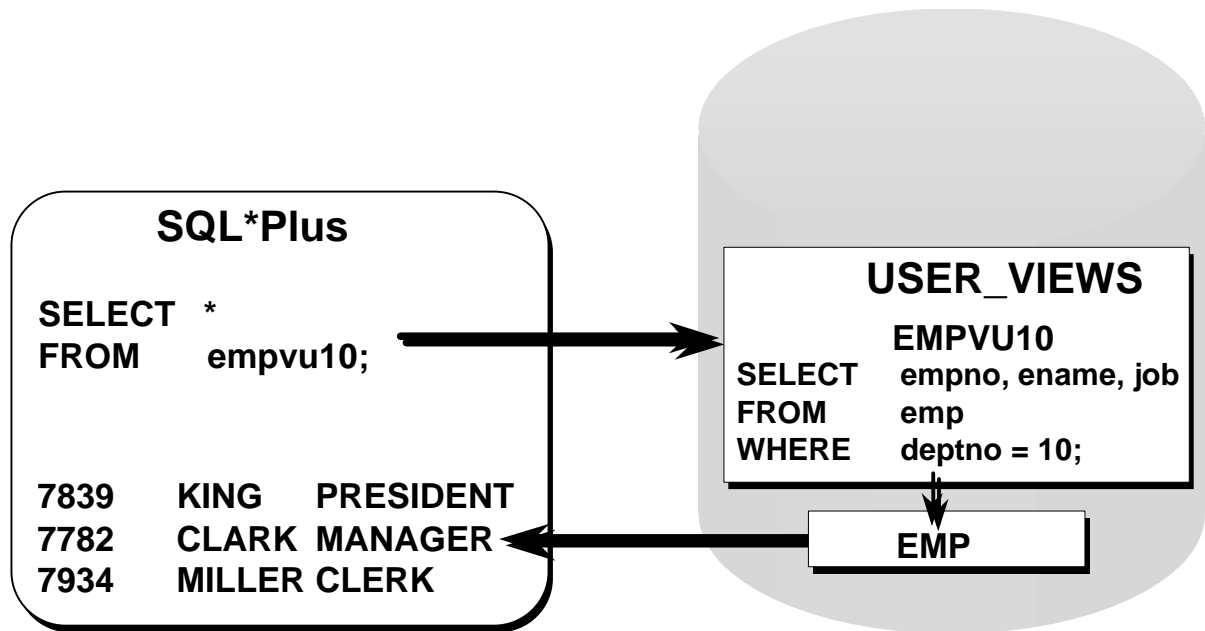
EMPLOYEE_NUMBER	NAME	SALARY
7698	BLAKE	2850
7654	MARTIN	1250
7499	ALLEN	1600
7844	TURNER	1500
7900	JAMES	950
7521	WARD	1250

6 rows selected.

Retrieving Data from a View

You can retrieve data from a view as you would from any table. You can either display the contents of the entire view or just view specific rows and columns.

Querying the USER_VIEWS Data Dictionary Table



11-37

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Views in the Data Dictionary

Once your view has been created, you can query the data dictionary table called `USER_VIEWS` to see the name of the view and the view definition. The text of the `SELECT` statement that constitutes your view is stored in a `LONG` column.

Views Data Access

When you access data, using a view, Oracle8 Server performs the following operations:

1. Retrieves the view definition from the data dictionary table `USER_VIEWS`.
2. Checks access privileges for the view base table.
3. Converts the view query into an equivalent operation on the underlying base table or tables. In other words, data is retrieved from, or an update made to, the base table(s).

Class Management Note

The view text is stored in a column of `LONG` data type. You may need to set `ARRAYSIZE` to a smaller value or increase the value of `LONG` to view the text.

Modifying a View

- **Modify the EMPVU10 view by using CREATE OR REPLACE VIEW clause. Add an alias for each column name.**

```
SQL> CREATE OR REPLACE VIEW empvu10
  2      (employee_number, employee_name, job_title)
  3  AS SELECT      empno, ename, job
  4  FROM            emp
  5  WHERE           deptno = 10;
View created.
```

- **Column aliases in the CREATE VIEW clause are listed in the same order as the columns in the subquery.**

Modifying a View

The OR REPLACE option allows a view to be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, recreating, and regranteeing object privileges.

Note: When assigning column aliases in the CREATE VIEW clause, remember that the aliases are listed in the same order as the columns in the subquery.

Class Management Note

The OR REPLACE option started with Oracle7. With earlier versions of Oracle, if the view needed to be changed, it had to be dropped and recreated.

Demo: *l12emp.sql*

Creating a Complex View

Create a complex view that contains group functions to display values from two tables.

```
SQL> CREATE VIEW      dept_sum_vu
  2                  (name, minsal, maxsal, avgsal)
  3  AS SELECT        d.dname, MIN(e.sal), MAX(e.sal),
  4                  AVG(e.sal)
  5  FROM              emp e, dept d
  6  WHERE             e.deptno = d.deptno
  7  GROUP BY         d.dname;
View created.
```

11-39

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Creating a Complex View

The example above creates a complex view of the department names, minimum salary, maximum salary, and average salary by the department. Note that alternative names have been specified for the view. This is a requirement if any column of the view is derived from a function or an expression.

You can view the structure of the view by using the SQL*Plus DESCRIBE command. Display the contents of the view by issuing a SELECT statement.

```
SQL> SELECT  *
  2  FROM      dept_sum_vu;
```

NAME	MINSAL	MAXSAL	AVGSAL
ACCOUNTING	1300	5000	2916.6667
RESEARCH	800	3000	2175
SALES	950	2850	1566.6667

Rules for Performing DML Operations on a View

- You can perform DML operations on simple views.
- You cannot remove a row if the view contains the following:
 - Group functions
 - A GROUP BY clause
 - The DISTINCT keyword

Performing DML Operations on a View

You can perform DML operations on data through a view provided those operations follow certain rules.

You can remove a row from a view unless it contains any of the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword

Rules for Performing DML Operations on a View

- You cannot modify data in a view if it contains:
 - Any of the conditions mentioned in the previous slide
 - Columns defined by expressions
 - The ROWNUM pseudocolumn
- You cannot add data if:
 - The view contains any of the conditions mentioned in the previous slide
 - There are NOT NULL columns in the base tables that are not selected by the view

11-41

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Performing DML Operations on a View

You can modify data in a view unless it contains any of the conditions mentioned in the previous slide and any of the following:


- Columns defined by expressions— for example, SALARY * 12
- The ROWNUM pseudocolumn

You can add data through a view unless it contains any of the above and there are NOT NULL columns, without a default value, in the base table that are not selected by the view. All required values must be present in the view. Remember that you are adding values directly into the underlying table *through* the view.

For more information, see

SQL Reference, Release 8.0, “CREATE VIEW.”

Oracle8 Server



Note: Oracle7.3 and Oracle8 allow you, with some restrictions, to modify views that involve joins. The restrictions for DML operations described above also apply to join views. Any UPDATE, INSERT, or DELETE statement on a join view can modify only one underlying base table. If at least one column in the subquery join has a unique index, then it may be possible to modify one base table in a join view. You can query USER_UPDATABLE_COLUMNS to see whether the columns in a join view are updatable.

Class Management Note

(For Using the WITH CHECK OPTION clause)

If the user does not supply a constraint name, the system assigns a name in the form SYS_Cn, where *n* is an integer that makes the constraint name unique within the system.

Demo: *l12vu1.sql*, *l12vu2.sql*

Using the WITH CHECK OPTION Clause

- You can ensure that DML on the view stays within the domain of the view by using the **WITH CHECK OPTION**.

```
SQL> CREATE OR REPLACE VIEW empvu20
  2  AS SELECT      *
  3  FROM    emp
  4  WHERE   deptno = 20
  5  WITH CHECK OPTION CONSTRAINT empvu20_ck;
View created.
```

- Any attempt to change the department number for any row in the view will fail because it violates the **WITH CHECK OPTION** constraint.

11-42

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Using the WITH CHECK OPTION Clause

It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.

The WITH CHECK OPTION clause specifies that INSERTS and UPDATES performed through the view are not allowed to create rows that the view cannot select, and therefore it allows integrity constraints and data validation checks to be enforced on data being inserted or updated.

If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, with the constraint name if that has been specified.

```
SQL> UPDATE empvu20
  2  SET    deptno = 22
  3  WHERE  empno = 7788;
UPDATE empvu20
      *
ERROR at line 1:
ORA-02291: integrity constraint (USER.EMP_DEPTNO_FK) violated -
parent key not found
```

Note: No rows are updated because if the department number were to change to 22, the view would no longer be able to see that employee. Therefore, with the WITH CHECK OPTION clause, the view can see only department 20 employees and does not allow the department number for those employees to be changed through the view.

Denying DML Operations

- You can ensure that no DML operations occur by adding the **WITH READ ONLY** option to your view definition.

```
SQL> CREATE OR REPLACE VIEW empvu10
  2      (employee_number, employee_name, job_title)
  3  AS SELECT      empno, ename, job
  4  FROM            emp
  5  WHERE           deptno = 10
  6  WITH READ ONLY;
View created.
```

- Any attempt to perform a DML on any row in the view will result in Oracle8 Server error **ORA-01732**.

11-43

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Denying DML Operations

You can ensure that no DML operations occur on your view by creating it with the **WITH READ ONLY** option. The example above modifies the EMPVU10 view to prevent any DML operations on the view.

Any attempts to remove a row from the view will result in an error.

```
SQL> DELETE FROM empvu10
  2  WHERE      employee_number = 7782;
DELETE FROM empvu10
      *
ERROR at line 1:
ORA-01752:Cannot delete from view without exactly one key-preserved
table
```

Removing a View

- **Remove a view without losing data because a view is based on underlying tables in the database.**

```
DROP VIEW view;
```

```
SQL> DROP VIEW empvu10;  
View dropped.
```

Removing a View

You use the DROP VIEW statement to remove a view. The statement removes the view definition from the database. Dropping views has no effect on the tables on which the view was based. Views or other applications based on deleted views become invalid. Only the creator or a user with the DROP ANY VIEW privilege can remove a view.

In the syntax:

view is the name of the view

Summary

- **A view is derived from data in other tables or other views.**
- **A view provides the following advantages:**
 - **Restricts database access**
 - **Simplifies queries**
 - **Provides data independence**
 - **Allows multiple views of the same data**
 - **Can be dropped without removing the underlying data**

11-45

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

What Is a View?

A view is based on a table or another view and acts as a window through which data on tables can be viewed or changed. A view does not contain data. The definition of the view is stored in the data dictionary. You can see the definition of the view in the USER_VIEWS data dictionary table.

Advantages of Views

- Restrict database access
- Simplify queries
- Provide data independence
- Allow multiple views of the same data
- Remove views without affecting the underlying data

View Options

- Can be a simple view based on one table
- Can be a complex view based on more than one table or can contain groups or functions
- Can be replaced if one of the same name exists
- Contain a check constraint
- Can be read-only

Practice Overview

- **Creating a simple view**
- **Creating a complex view**
- **Creating a view with a check constraint**
- **Attempting to modify data in the view**
- **Displaying view definitions**
- **Removing views**

Practice Overview

In this practice, you will create simple and complex views, and attempt to perform DML statements on the views.

Practice 12

1. Create a view called EMP_VU based on the employee number, employee name, and department number from the EMP table. Change the heading for the employee name to EMPLOYEE.
2. Display the content's of the EMP_VU view.

```
EMPNO  EMPLOYEE  DEPTNO
```

```
-----
7839 KING      10
7698 BLAKE     30
7782 CLARK     10
7566 JONES     20
7654 MARTIN    30
7499 ALLEN     30
7844 TURNER    30
7900 JAMES     30
7521 WARD      30
7902 FORD      20
7369 SMITH     20
7788 SCOTT     20
7876 ADAMS     20
7934 MILLER    10
14 rows selected.
```

3. Select the view_name and text from the data dictionary USER_VIEWS.

```
VIEW_NAME  TEXT
-----
EMP_VU     SELECT empno, ename employee, deptno
          FROM emp
```

4. Using your view EMP_VU, enter a query to display all employee names and department numbers.

```
EMPLOYEE      DEPTNO
-----
KING          10
BLAKE         30
```

```
CLARK         10
JONES         20
```

```
MARTIN        30
```

```
...
```

```
14 rows selected.
```

Practice 12 (continued)

5. Create a view named DEPT20 that contains the employee number, employee name, and department number for all employees in department 20. Label the view column EMPLOYEE_ID, EMPLOYEE, and DEPARTMENT_ID. Do not allow an employee to be reassigned to another department through the view.
6. Display the structure and contents of the DEPT20 view.

Name	Null?	Type
=====	=====	=====
EMPLOYEE_ID	NOT NULL	NUMBER (4)
EMPLOYEE		VARCHAR2 (10)
DEPARTMENT_ID	NOT NULL	NUMBER (2)

EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID

7566 JONES		20
7902 FORD		20
7369 SMITH		20
7788 SCOTT		20
7876 ADAMS		20

7. Attempt to reassign Smith to department 30.

If you have time, complete the following exercises.

8. Create a view called SALARY_VU based on the employee name, department name, salary and salary grade for all employees. Label the columns Employee, Department, Salary and Grade, respectively.

11

Other Database Objects

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	20 minutes	Lecture
	15 minutes	Practice
	35 minutes	Total

Objectives

At the end of this lesson, you will be able to:

- **Describe some database objects and their uses**
- **Create, maintain, and use sequences**
- **Create and maintain indexes**
- **Create private and public synonyms**

Lesson Aim

In this lesson, you will learn how to create and maintain some of the other commonly used database objects. These objects include sequences, indexes, and synonyms.

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence	Generates primary key values
Index	Improves the performance of some queries
Synonym	Alternative name for an object

Database Objects

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.

If you want to improve the performance of some queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using synonyms.

What Is a Sequence?

- **Automatically generates unique numbers**
- **Is a sharable object**
- **Is typically used to create a primary key value**
- **Replaces application code**
- **Speeds up the efficiency of accessing sequence values when cached in memory**

What Is a Sequence?

A sequence generator can be used to automatically generate sequence numbers for rows in tables. A sequence is a database object created by a user and can be shared by multiple users.

A typical usage for sequences is to create a primary key value, which must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle8 routine. This can be a time-saving object because it can reduce the amount of application code needed to write a sequence generating routine.

Sequence numbers are stored and generated independently of tables. Therefore, the same sequence can be used for multiple tables.

The CREATE SEQUENCE Statement

Define a sequence to generate sequential numbers automatically

```
CREATE SEQUENCE sequence
    [INCREMENT BY n]
    [START WITH n]
    [{MAXVALUE n | NOMAXVALUE} ]
    [{MINVALUE n | NOMINVALUE} ]
    [{CYCLE | NOCYCLE} ]
    [{CACHE n | NOCACHE} ] ;
```

11-53

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Creating a Sequence

Automatically generate sequential numbers by using the CREATE SEQUENCE statement.

In the syntax:

<i>sequence</i>	is the name of the sequence generator.
INCREMENT BY <i>n</i>	specifies the interval between sequence numbers where <i>n</i> is an integer. If this clause is omitted, the sequence will increment by 1.
START WITH <i>n</i>	specifies the first sequence number to be generated. If this clause is omitted, the sequence will start with 1.
MAXVALUE <i>n</i>	specifies the maximum value the sequence can generate.
NOMAXVALUE	specifies a maximum value of 10^{27} for an ascending sequence and -1 for a descending sequence. This is the default option.
MINVALUE <i>n</i>	specifies the minimum sequence value.
NOMINVALUE	specifies a minimum value of 1 for an ascending sequence and $-(10^{26})$ for a descending sequence. This is the default option.
CYCLE NOCYCLE	specifies that the sequence continues to generate values after reaching either its maximum or minimum value or does not generate additional values. NOCYCLE is the default option.
CACHE <i>n</i> NOCACHE	specifies how many values the Oracle8 Server will preallocate and keep in memory. By default, the Oracle8 Server will cache 20 values.

Creating a Sequence

- Create a sequence named DEPT_DEPTNO to be used for the primary key of the DEPT table.
- Do not use the CYCLE option.

```
SQL> CREATE SEQUENCE dept_deptno
      2      INCREMENT BY 1
      3      START WITH 91
      4      MAXVALUE 100
      5      NOCACHE
      6      NOCYCLE;
Sequence created.
```

11-54

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Creating a Sequence

The example above creates a sequence named DEPT_DEPTNO to be used for the DEPTNO column of the DEPT table. The sequence starts at 91, does not allow caching, and does not allow the sequence to cycle.

Do not use the CYCLE option if the sequence is used to generate primary key values unless you have a reliable mechanism that purges old rows faster than the sequence cycles.



For more information, see

Oracle8 Server SQL Reference, Release 8.0, "CREATE SEQUENCE."



Class Management Note

If the INCREMENT BY value is negative, the sequence will descend.

Also, ORDER |

NOORDER options are available. The ORDER option guarantees that sequence values are generated in order. It is not important if you use the sequence to generate primary key values. This option is relevant only with the Parallel Server option.

If sequence values are cached, they will be lost if there is a system failure.

Confirming Sequences

- Verify your sequence values in the **USER_SEQUENCES** data dictionary table.

```
SQL> SELECT  sequence_name, min_value, max_value,
2           increment_by, last_number
3 FROM      user_sequences;
```

- The **LAST_NUMBER** column displays the next available sequence number.

11-55

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Confirming Sequences

Once you have created your sequence, it is documented in the data dictionary. Since a sequence is a database object, you can identify it in the **USER_OBJECTS** data dictionary table.

You can also confirm the settings of the sequence by selecting from the data dictionary's **USER_SEQUENCES** table.

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
---------------	-----------	-----------	--------------	-------------

CUSTID	1	1.000E+27	1	109
DEPT_DEPTNO	1	100	1	91
ORDID	1	1.000E+27	1	622
PRODID	1	1.000E+27	1	200381

Class Management Note

Demo: *l13dd.sql*

NEXTVAL and CURRVAL Pseudocolumns

- **NEXTVAL** returns the next available sequence value.

It returns a unique value every time it is referenced, even for different users.

- **CURRVAL** obtains the current sequence value.

NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Using a Sequence

Once you create your sequence, you can use the sequence to generate sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

NEXTVAL and CURRVAL Pseudocolumns

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. You must qualify NEXTVAL with the sequence name. When you reference *sequence*.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced. You must qualify CURRVAL with the sequence name. When *sequence*.CURRVAL is referenced, the last value returned to that user's process is displayed.

NEXTVAL and CURRVAL Pseudocolumns

- **NEXTVAL** returns the next available sequence value.

It returns a unique value every time it is referenced, even for different users.

- **CURRVAL** obtains the current sequence value.

NEXTVAL must be issued for that sequence before CURRVAL contains a value.

11-57

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Rules for Using NEXTVAL and CURRVAL

You can use NEXTVAL and CURRVAL in the following:

- The SELECT list of a SELECT statement that is not part of a subquery
- The SELECT list of a subquery in an INSERT statement
- The VALUES clause of an INSERT statement
- The SET clause of an UPDATE statement

You cannot use NEXTVAL and CURRVAL in the following:

- A SELECT list of a view
- A SELECT statement with the DISTINCT keyword
- A SELECT statement with the GROUP BY, HAVING, or ORDER BY clauses
- A subquery in a SELECT, DELETE, or UPDATE statement
- A DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

For more information, see

Oracle8 Server SQL Reference, Release 8.0, “Pseudocolumns” section and “CREATE SEQUENCE.”



Class Management Note

Be sure to point out the rules listed on this page.

Using a Sequence

- Insert a new department named “MARKETING” in San Diego.

```
SQL> INSERT INTO      dept(deptno, dname, loc)
  2  VALUES          (dept_deptno.NEXTVAL,
  3                    'MARKETING', 'SAN DIEGO');
1 row created.
```

- View the current value for the DEPT_DEPTNO sequence.

```
SQL> SELECT  dept_deptno.CURRVAL
  2  FROM      SYS.dual;
```

11-58

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Using a Sequence

The example on the slide inserts a new department in the DEPT table. It uses the DEPT_DEPTNO sequence for generating a new department number.

You can view the current value of the sequence:

```
SQL> SELECT  dept_deptno.CURRVAL
  2  FROM      SYS.dual;
```

CURRVAL

Suppose now you want to hire employees to staff the new department. The INSERT statement that can be executed repeatedly for all the new employees can include the following code:

Note: The above example assumes that a sequence EMP_EMPNO has already been created for generating a new employee number.

```
SQL> INSERT INTO emp ...
  2  VALUES (emp_empno.NEXTVAL, dept_deptno.CURRVAL, ...
```

Using a Sequence

- **Caching sequence values in memory allows faster access to those values.**
- **Gaps in sequence values can occur when:**
 - **A rollback occurs**
 - **The system crashes**
 - **A sequence is used in another table**
- **View the next available sequence, if it was created with NOCACHE, by querying the USER_SEQUENCES table.**

Caching Sequence Values

Cache sequences in the memory to allow faster access to those sequence values. The cache is populated at the first reference to the sequence. Each request for the next sequence value is retrieved from the cached sequence. After the last sequence is used, the next request for the sequence pulls another cache of sequences into memory.

Beware of Gaps in Your Sequence

Although sequence generators issue sequential numbers without gaps, this action occurs independent of a commit or rollback. Therefore, if you roll back a statement containing a sequence, the number is lost.

Another event that can cause gaps in the sequence is a system crash. If the sequence caches values in the memory, then those values are lost if the system crashes.

Because sequences are not tied directly to tables, the same sequence can be used for multiple tables. If this occurs, each table can contain gaps in the sequential numbers.

Viewing the Next Available Sequence Value Without Incrementing It

It is possible to view the next available sequence value without incrementing it, only if the sequence was created with NOCACHE, by querying the USER_SEQUENCES table.

Class Management Note

Frequently used sequences should be created with caching to improve efficiency. For cached sequences, there is no way to find out what the next available sequence value will be without actually obtaining, and using up, that value. It is recommended that users resist finding the next sequence value. Trust the system to provide a unique value each time a sequence is used in an INSERT statement.

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option.

```
SQL> ALTER SEQUENCE dept_deptno
      2      INCREMENT BY 1
      3      MAXVALUE 999999
      4      NOCACHE
      5      NOCYCLE;
Sequence altered.
```

11-60

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Altering a Sequence

If you reach the MAXVALUE limit for your sequence, no additional values from the sequence will be allocated and you will receive an error indicating that the sequence exceeds the MAXVALUE. To continue to use the sequence, you can modify it by using the ALTER SEQUENCE statement.

Syntax

```
ALTER SEQUENCE sequence
  [ INCREMENT BY n ]
  [ { MAXVALUE n | NOMAXVALUE } ]
  [ { MINVALUE n | NOMINVALUE } ]
  [ { CYCLE | NOCYCLE } ]
  [ { CACHE n | NOCACHE } ] ;
```

where: *sequence* is the name of the sequence generator

For more information, see

Oracle8 Server SQL Reference, Release 8.0, "ALTER SEQUENCE."



Guidelines for Modifying a Sequence

- You must be the owner or have the **ALTER** privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.

11-61

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Guidelines

- You must own or you have the ALTER privilege for the sequence in order to modify it.
- Only future sequence numbers are affected by the ALTER SEQUENCE statement.
- The START WITH option cannot be changed using ALTER SEQUENCE. The sequence must be dropped and re-created in order to restart the sequence at a different number.
- Some validation is performed. For example, a new MAXVALUE cannot be imposed that is less than the current sequence number.

```
SQL> ALTER SEQUENCE dept_deptno
  2      INCREMENT BY 1
  3      MAXVALUE 90
  4      NOCACHE
  5      NOCYCLE;
ALTER SEQUENCE dept_deptno
*
ERROR at line 1:
ORA-04009: MAXVALUE cannot be made to be less than the current
value
```

Removing a Sequence

- Remove a sequence from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the sequence can no longer be referenced.

```
SQL> DROP SEQUENCE dept_deptno;  
Sequence dropped.
```

Removing a Sequence

To remove a sequence from the data dictionary, use the DROP SEQUENCE statement. You must be the owner of the sequence or have the DROP ANY SEQUENCE privilege to remove it.

Syntax

```
DROP SEQUENCE sequence;
```

where: *sequence* is the name of the sequence generator

For more information, see

Oracle8 Server SQL Reference, Release 8.0, "DROP SEQUENCE."



What Is an Index?

- **Schema object**
- **Used by the Oracle8 Server to speed up the retrieval of rows by using a pointer**
- **Reduces disk I/O by using rapid path access method to locate the data quickly**
- **Independent of the table it indexes**
- **Automatically used and maintained by the Oracle8 Server**

11-63

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

What Is an Index?

An Oracle8 Server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If you do not have an index on the column, then a full table scan will occur.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is automatically used and maintained by the Oracle8 Server. Once an index is created, no direct activity is required by the user.

Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

Note: When you drop a table, corresponding indexes are also dropped.

For more information, see

Oracle8 Server Concepts Manual, Release 8.0, “Schema Objects” section, “Indexes” topic.

Class Management Note



The decision to create indexes is a global, high-level decision. Creation and maintenance of indexes is often a DBA task.

Reference the column that has an index in the predicate WHERE clause without modifying the indexed column with a function or expression.

A ROWID is a hexadecimal string representation of the row address containing block identifier, row location in the block, and the database file identifier. The fastest way to access any particular row is by referencing its ROWID.

How Are Indexes Created?

- **Automatically**
 - **A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.**
- **Manually**
 - **Users can create nonunique indexes on columns to speed up access time to the rows.**

How Are Indexes Created?

Two types of indexes can be created. One type is a unique index. The Oracle8 Server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE constraint. The name of the index is the name given to the constraint.

The other type of index a user can create is a nonunique index. For example, you can create a FOREIGN KEY column index for a join in a query to improve retrieval speed.

Creating an Index

- Create an index on one or more columns

```
CREATE INDEX index  
ON table (column[, column]...);
```

- Improve the speed of query access on the ENAME column in the EMP table

```
SQL> CREATE INDEX      emp_ename_idx  
2  ON                  emp(ename);  
Index created.
```

Creating an Index

Create an index on one or more columns by issuing the CREATE INDEX statement.

In the syntax:

<i>index</i>	is the name of the index
<i>table</i>	is the name of the table
<i>column</i>	is the name of the column in the table to be indexed

For more information, see

Oracle8 Server SQL Reference, Release 8.0, "CREATE INDEX."



Class Management Note

To create an index in your schema, you must have CREATE INDEX privileges.

Another option in the syntax is the UNIQUE keyword. Emphasize that Oracle recommends that you do not explicitly define unique indexes on tables. Instead define uniqueness in the table as a constraint. Oracle8 Server enforces unique integrity constraints by automatically defining a unique index on the unique key.

Guidelines to Creating an Index

- The column is used frequently in the WHERE clause or in a join condition.
- The column contains a wide range of values.
- The column contains a large number of null values.
- Two or more columns are frequently used together in a WHERE clause or a join condition.
- The table is large and most queries are expected to retrieve less than 2–4% of the rows.

11-66

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

More Is Not Always Better

More indexes on a table does not mean it will speed up queries. Each DML operation that is committed on a table with indexes means that the indexes must be updated. The more indexes you have associated with a table, the more effort the Oracle8 Server must make to update all the indexes after a DML.

When to Create an Index

- The column is used frequently in the WHERE clause or in a join condition.
- The column contains a wide range of values.
- The column contains a large number of null values.
- Two or more columns are frequently used together in a WHERE clause or join condition.
- The table is large and most queries are expected to retrieve less than 2-4% of the rows.

Remember that if you want to enforce uniqueness, you should define a unique constraint in the table definition. Then, a unique index is automatically created.



Guidelines to Creating an Index

Do not create an index if:

- **The table is small**
- **The columns are not often used as a condition in the query**
- **Most queries are expected to retrieve more than 2–4% of the rows**
- **The table is updated frequently**

When to Not Create an Index

- The table is small.
- The columns are not often used as a condition in the query.
- Most queries are expected to retrieve more than 2-4% of the rows.
- The table is updated frequently. If you have one or more indexes on a table, the DML statements that access the table take relatively more time due to maintenance of indexes.

Class Management Note

Null values are not included in the index.

To optimize joins, you can create an index on the FOREIGN KEY column, which will speed up the search to match rows to the PRIMARY KEY column.

The optimizer does not use an index if the WHERE clause contains the IS NULL expression.

Confirming Indexes

- The **USER_INDEXES** data dictionary view contains the name of the index and its uniqueness.
- The **USER_IND_COLUMNS** view contains the index name, the table name, and the column name.

```
SQL> SELECT  ic.index_name, ic.column_name,
2           ic.column_position col_pos, ix.uniqueness
3 FROM      user_indexes ix, user_ind_columns ic
4 WHERE     ic.index_name = ix.index_name
5 AND       ic.table_name = 'EMP';
```

11-68

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Confirming Indexes

Confirm the existence of indexes from the **USER_INDEXES** data dictionary view. You can also check the columns involved in an index by querying the **USER_IND_COLUMNS** view.

The example above displays all the previously created indexes, affected column names, and uniqueness on the **EMP** table.

INDEX_NAME	COLUMN_NAME	COL_POS	UNIQUENES
-----	-----	-----	-----
EMP_EMPNO_PK	EMPNO	1	UNIQUE
EMP_ENAME_IDX	ENAME	1	NONUNIQUE

Note: The EMP table has been formatted.

Removing an Index

- Remove an index from the data dictionary.

```
SQL> DROP INDEX index;
```

- Remove the EMP_ENAME_IDX index from the data dictionary.

```
SQL> DROP INDEX emp_ename_idx;  
Index dropped.
```

- To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

11-69

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Removing an Index

You cannot modify indexes. To change an index, you must drop it and then re-create it. Remove an index definition from the data dictionary by issuing the DROP INDEX statement. To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

In the syntax:

index is the name of the index

Class Management Note

Remind students that if they drop a table, indexes and constraints are automatically dropped, but views and sequences remain.

Synonyms

Simplify access to objects by creating a synonym (another name for an object).

- **Refer to a table owned by another user.**
- **Shorten lengthy object names.**

```
CREATE [PUBLIC] SYNONYM synonym  
FOR      object;
```

11-70

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Creating a Synonym for an Object

To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other objects. This method can be especially useful with lengthy object names, such as views.

In the syntax:

PUBLIC	creates a synonym accessible to all users
<i>synonym</i>	is the name of the synonym to be created
<i>object</i>	identifies the object for which the synonym is created

Guidelines

- The object cannot be contained in a package.
- A private synonym name must be distinct from all other objects owned by the same user.

For more information, see

Oracle8 Server SQL Reference, Release 8.0, "CREATE SYNONYM."



Creating and Removing Synonyms

- Create a shortened name for the DEPT_SUM_VU view.

```
SQL> CREATE SYNONYM d_sum  
2 FOR dept_sum_vu;  
Synonym Created.
```

- Drop a synonym.

```
SQL> DROP SYNONYM d_sum;  
Synonym dropped.
```

11-71

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Creating a Synonym for an Object

The example above creates a synonym for the DEPT_SUM_VU view for quicker reference.

The DBA can create a public synonym accessible to all users. The example below creates a public synonym named DEPT for Alice's DEPT table:

```
SQL> CREATE PUBLIC SYNONYM dept  
2 FOR alice.dept;  
Synonym created.
```

Removing a Synonym

To drop a synonym, use the DROP SYNONYM statement. Only the DBA can drop a public synonym.

```
SQL> DROP SYNONYM dept;  
Synonym dropped.
```

For more information, see
Oracle8 Server SQL Reference, Release 8.0, "DROP SYNONYM."

Class Management Note

In the Oracle8 Server, the DBA can specifically grant the CREATE PUBLIC SYNONYM to any user, allowing that user to create public synonyms.



Summary

- **Automatically generate sequence numbers by using a sequence generator.**
- **View sequence information in the USER_SEQUENCES data dictionary table.**
- **Create indexes to improve query retrieval speed.**
- **View index information in the USER_INDEXES dictionary table.**
- **Use synonyms to provide alternative names for objects.**

Sequences

The sequence generator can be used to automatically generate sequence numbers for rows in tables. This can be time saving and can reduce the amount of application code needed.

A sequence is a database object that can be shared with other users. Information about the sequence can be found in the USER_SEQUENCES table of the data dictionary.

To use a sequence, reference it with either the NEXTVAL or the CURRVAL pseudocolumns.

- Retrieve the next number in the sequence by referencing *sequence*.NEXTVAL.
- Return the current available number by referencing *sequence*.CURRVAL.

Indexes

Indexes are used to improve the query retrieval speed.

Users can view the definitions of the indexes in the USER_INDEXES data dictionary view.

An index can be dropped by the creator or a user with the DROP ANY INDEX privilege by using the DROP INDEX statement.

Synonyms

DBAs can create public synonyms, and users can create private synonyms for convenience by using the CREATE SYNONYM statement. Synonyms permit short names or alternative names for objects. Remove synonyms by using the DROP SYNONYM statement.

Practice Overview

- **Creating sequences**
- **Using sequences**
- **Creating nonunique indexes**
- **Display data dictionary information about sequences and indexes**
- **Dropping indexes**

11-73

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Practice Overview

In this practice, you will create a sequence to be used when populating your DEPARTMENT table. You will also create implicit and explicit indexes.

Practice 13

1. Create a sequence to be used with the DEPARTMENT table's primary key column. The sequence should start at 60 and have a maximum value of 200. Have your sequence increment by ten numbers. Name the sequence DEPT_ID_SEQ.
2. Write a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script *p13q2.sql*. Execute your script.

SEQUENCE_NAME	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
CUSTID	1.000E+27	1	109
DEPT_ID_SEQ	200	1	60
ORDID	1.000E+27	1	622
PRODID	1.000E+27	1	200381

3. Write an interactive script to insert a row into the DEPARTMENT table. Name your script *p13q3.sql*. Be sure to use the sequence that you created for the ID column. Create a customized prompt to enter the department name. Execute your script. Add two departments named Education and Administration. Confirm your additions.
4. Create a non-unique index on the FOREIGN KEY column in the EMPLOYEE table.
5. Display the indexes and uniqueness that exist in the data dictionary for the EMPLOYEE table. Save the statement into a script named *p13q5.sql*.

INDEX_NAME	TABLE_NAME	UNIQUENES
EMPLOYEE_DEPT_ID_IDX	EMPLOYEE	NONUNIQUE
EMPLOYEE_ID_PK	EMPLOYEE	UNIQUE

6. ~~Create a PRIMARY KEY constraint on the DEPARTMENT table. Confirm the constraint~~ in the data dictionary by executing *p11_q3.sql*. ~~Modify and Confirm the unique index in the data dictionary by executing *p13q5.sql*.~~

CONSTRAINT_NAME	C
DEPARTMENT_ID_PK	P
EMPLOYEE_ID_PK	P
EMPLOYEE_DEPT_ID_FK	R

INDEX_NAME	TABLE_NAME	UNIQUENES
DEPARTMENT_ID_PK	DEPARTMENT	UNIQUE
EMPLOYEE_DEPT_ID_IDX	EMPLOYEE	NONUNIQUE
EMPLOYEE_ID_PK	EMPLOYEE	UNIQUE

11

Controlling User Access

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	15 minutes	Lecture
	20 minutes	Practice
	35 minutes	Total

Objectives

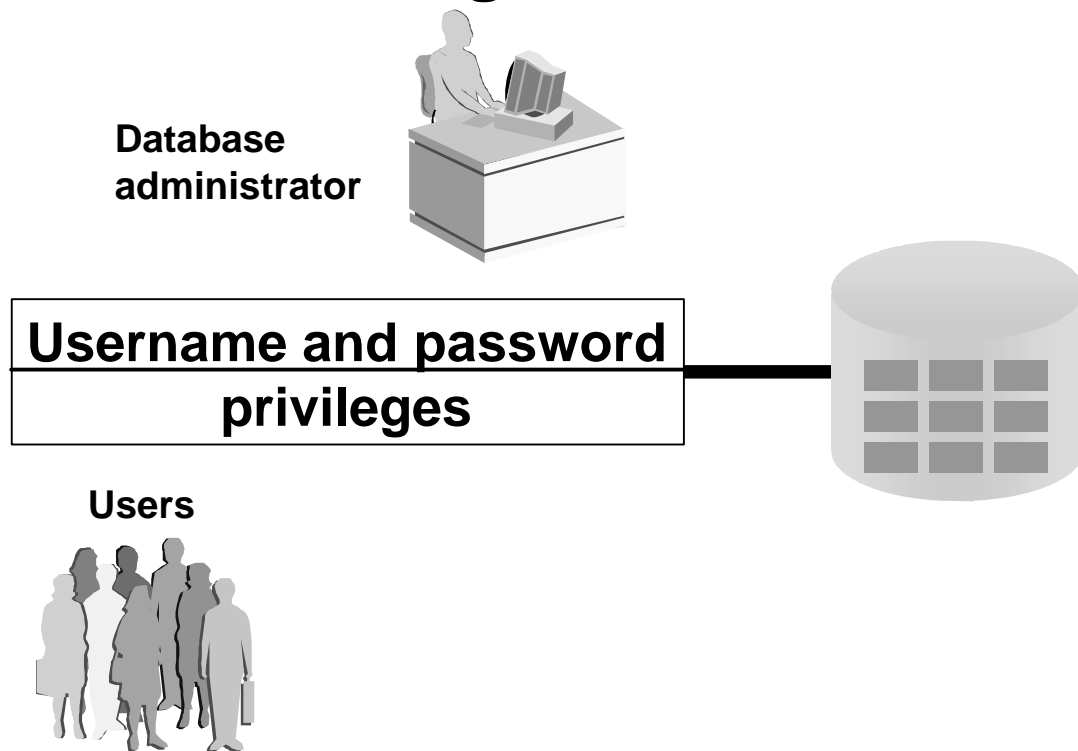
At the end of this lesson, you should be able to:

- **Create users**
- **Create roles to ease setup and maintenance of the security model**
- **GRANT and REVOKE object privileges**

Lesson Aim

In this lesson, you will learn how to control database access to specific objects and add new users with different levels of access privileges.

Controlling User Access



11-77

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Controlling User Access

In a multiple-user environment, you want to maintain security of the database access and use. Oracle8 Server database security allows you to do the following:

- Control database access
- Give access to specific objects in the database
- Confirm given and received *privileges* with the Oracle data dictionary
- Create synonyms for database objects

Database security can be classified into two categories: system security and data security. System security covers access and use of the database at the system level, such as username and password, disk space allocated to users, and system operations allowed by the user. Database security covers access and use of the database objects and the actions that those users can have on the objects.

Privileges

- **Database security**
 - **System security**
 - **Data security**
- **System privileges: Gain access to the database**
- **Object privileges: Manipulate the content of the database objects**
- **Schema: Collection of objects, such as tables, views, and sequences**

11-78

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Privileges

Privileges are the right to execute particular SQL statements. The database administrator is a high-level user with the ability to grant users access to the database and its objects. The users require *system privileges* to gain access to the database and *object privileges* to manipulate the content of the objects in the database. Users can also be given the privilege to grant additional privileges to other users or to *roles*, which are named groups of related privileges.

Schema

A **schema** is a collection of objects, such as tables, views, and sequences. The schema is owned by a database user and has the same name as that user.

For more information, see *Oracle8 Server Application Developer's Guide, Release 8.0*, "Establishing a Security Policy" section, and *Oracle8 Server Concepts Manual, Release 8.0*, "Database Security" topic.



System Privileges

- More than 80 privileges are available.
- The DBA has high-level system privileges.
 - Create new users
 - Remove users
 - Remove tables
 - Backup tables

System Privileges

More than 80 system privileges are available for users and roles. System privileges are typically provided by the database administrator.

Typical DBA Privileges

System Privilege	Operations Authorized
CREATE USER	Allows grantee to create other Oracle users (a privilege required for a DBA role)
DROP USER	Drops another user
DROP ANY TABLE	Drops a table in any schema
BACKUP ANY TABLE	Backs up any table in any schema with the export utility

Creating Users

The DBA creates users by using the **CREATE USER** statement.

```
CREATE USER      user
IDENTIFIED BY    password;
```

```
SQL> CREATE USER scott
      2 IDENTIFIED BY tiger;
User created.
```

11-80

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Creating a User

The DBA creates the user by executing the **CREATE USER** statement. The user does not have any privileges at this point. The DBA can then grant a number of privileges to that user. These privileges determine what the user can do at the database level.

The slide gives the abridged syntax for creating a user. In the syntax:

user is the name of the user to be created
password specifies that the user must log in with this password

For more information, see *Oracle8 Server SQL Reference, Release 8.0*, “GRANT” (System Privileges and Roles) and “CREATE USER.”



User System Privileges

- Once a user is created, the DBA can grant specific system privileges to a user.

```
GRANT privilege [, privilege...]  
TO user [, user...];
```

- An application developer has the following system privileges:
 - CREATE SESSION
 - CREATE TABLE
 - CREATE SEQUENCE
 - CREATE VIEW
 - CREATE PROCEDURE

11-81

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Typical User Privileges

Now that the DBA has created a user, the DBA can assign privileges to that user.

System Privilege	Operations Authorized
CREATE SESSION	Connect to the database
CREATE TABLE	Create tables in the user's schema
CREATE SEQUENCE	Create a sequence in the user's schema
CREATE VIEW	Create a view in the user's schema
CREATE PROCEDURE	Create a stored procedure, function, or package in the user's schema

In the syntax:

privilege

is the system privilege to be granted

user

is the name of the user

Granting System Privileges

The DBA can grant a user specific system privileges.

```
SQL> GRANT  create table, create sequence, create view  
3  TO      scott;  
Grant succeeded.
```

Granting System Privileges

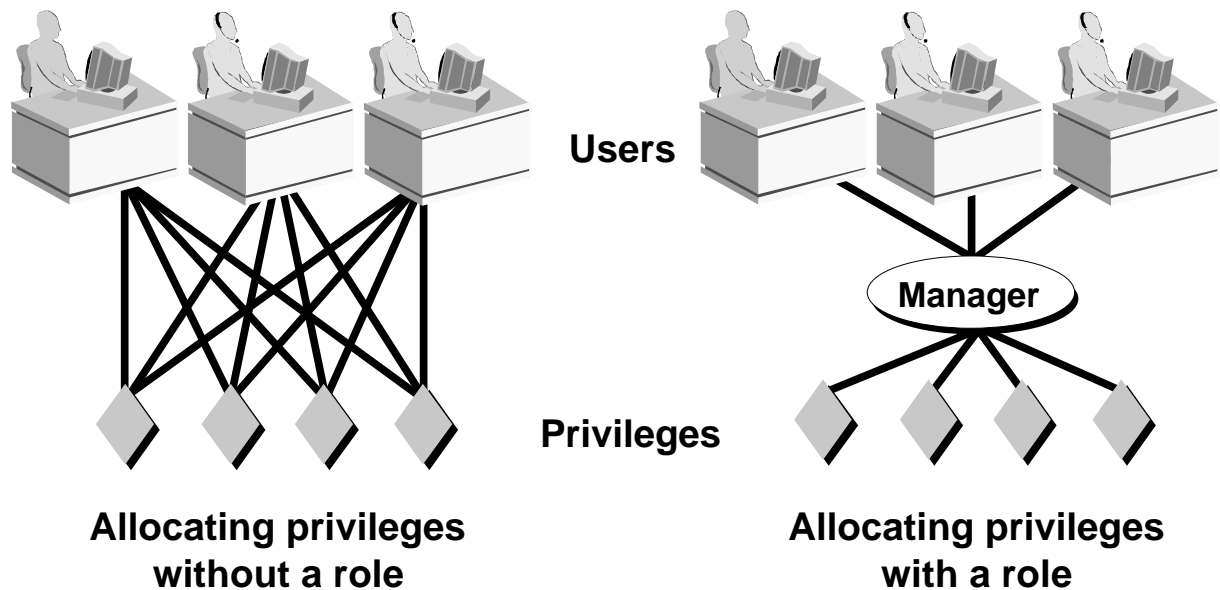
The DBA uses the GRANT statement to allocate system privileges to the user. Once the user has been granted the privileges, the user can immediately use those privileges.

In the above example, user Scott has been assigned the privileges to create tables, sequences, and views.

Class Management Note

User needs to have the required space quota to create tables.

What Is a Role?



11-83

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

What Is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes granting and revoking privileges easier to perform and maintain.

A user can have access to several roles, and several users can be assigned the same role. Roles typically are created for a database application.

Creating and Assigning a Role

First, the DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax

```
CREATE    ROLE  role;
```

where: *role* is the name of the role to be created

Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.

Class Management Note

Discuss the four following points about roles:

- Named groups of related privileges
- Can be granted to users
- Simplifies the process of granting and revoking privileges
- Created by a DBA

Creating a Role

```
SQL> CREATE ROLE manager;  
Role created.
```

```
SQL> GRANT create table, create view  
2          to manager;  
Grant succeeded.
```

```
SQL> GRANT manager to BLAKE, CLARK;  
Grant succeeded.
```

Creating a Role

The example above creates a role manager and then allows the managers to create tables and views. It then grants Blake and Clark the role of managers. Now Blake and Clark can create tables and views.

Changing Your Password

- When the user account is created, a password is initialized.
- Users can change their password by using the **ALTER USER** statement.

```
SQL> ALTER USER scott  
2 IDENTIFIED BY lion;  
User altered.
```

Changing Your Password

Every user has a password that is initialized by the DBA when the user is created. You can change your password by using the ALTER USER statement.

Syntax

```
ALTER USER user IDENTIFIED BY password;
```

where: *user* is the name of the user
password specifies the new password

Although this statement can be used to change your password, there are many other options. You must have the ALTER USER privilege to change any other option.

For more information, see *Oracle8 Server SQL Reference, Release 8.0*, "ALTER USER."



Object Privileges

Object Privilege	Table	View	Sequence	Procedure
ALTER	Ö		Ö	
DELETE	Ö	Ö		
EXECUTE				Ö
INDEX	Ö			
INSERT	Ö	Ö		
REFERENCES	Ö			
SELECT	Ö	Ö	Ö	
UPDATE	Ö	Ö		

Object Privileges

An *object privilege* is a privilege or right to perform a particular action on a specific table, view, sequence, procedure, function, or package. Each object has a particular set of grantable privileges. The table above lists the privileges for various objects. Note that the only privileges that apply to a sequence are SELECT and ALTER. UPDATE, REFERENCES, and INSERT can be restricted by specifying a subset of updatable columns. A SELECT can be restricted by creating a view with a subset of columns and granting SELECT privilege on the view. A grant on a synonym is converted to a grant on the base table referenced by the synonym.

Note: A procedure refers to standalone procedures and functions and to public package constructs. The INDEX and REFERENCES privileges cannot be granted to a role.

Object Privileges

- Object privileges vary from object to object.
- An owner has all the privileges on the object.
- An owner can give specific privileges on that owner's object.

```
GRANT      object_priv [(columns)]  
ON         object  
TO         {user|role|PUBLIC}  
[WITH GRANT OPTION];
```

Granting Object Privileges

Different object privileges are available for different types of schema objects. A user automatically has all object privileges for schema objects contained in his or her schema. A user can grant any object privilege on any schema object he or she owns to any other user or role. If the grant includes the GRANT OPTION, the grantee can further grant the object privilege to other users; otherwise, the grantee can use the privilege but cannot grant it to other users.

In the syntax:

<i>object_priv</i>	is an object privilege to be granted	
ALL	all object privileges	
<i>columns</i>	specifies the column from a table or view on which privileges	are
granted		
ON <i>object</i>	is the object on which the privileges are granted	
TO	identifies to whom the privilege is granted	
PUBLIC	grants object privileges to all users	
WITH GRANT OPTION	allows the grantee to grant the object privileges to other users	
and roles		

Granting Object Privileges

- Grant query privileges on the EMP table.

```
SQL> GRANT      select
  2  ON          emp
  3  TO          sue, rich;
Grant succeeded.
```

- Grant privileges to update specific columns to users and roles.

```
SQL> GRANT      update (dname, loc)
  2  ON          dept
  3  TO          scott, manager;
Grant succeeded.
```

Guidelines

- To grant privileges on an object, the object must be in your own schema or you must have been granted the object privileges WITH GRANT OPTION.
- An object owner can grant any object privilege on the object to any other user or role of the database.
- The owner of an object automatically acquires all object privileges on that object.

The first example above grants users Sue and Rich the privilege to query your EMP table. The second example grants UPDATE privileges on specific columns in the DEPT table to Scott and to the manager role.

Note: DBAs generally allocate system privileges; any user who owns an object can grant object privileges.

Using WITH GRANT OPTION and PUBLIC Keywords

- Give a user authority to pass along the privileges.

```
SQL> GRANT      select, insert
  2  ON          dept
  3  TO          scott
  4  WITH GRANT OPTION;
Grant succeeded.
```

- Allow all users on the system to query data from Alice's DEPT table.

```
SQL> GRANT      select
  2  ON          alice.dept
  3  TO          PUBLIC;
Grant succeeded.
```

11-89

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

WITH GRANT OPTION Keyword

A privilege that is granted WITH GRANT OPTION can be passed on to other users and roles by the grantee. Object privileges granted WITH GRANT OPTION are revoked when the grantor's privilege is revoked.

The above example allows user Scott to access your DEPT table with the privileges to query the table and add rows to the table. Allow Scott to give others these privileges.

PUBLIC Keyword

An owner of a table can grant access to all users by using the PUBLIC keyword.

The above example allows all users on the system to query data from Alice's DEPT table.

Class Management Note

If a statement does not use the full name of an object, the Oracle8 Server implicitly prefixes the object name with the current user's name (or schema). If user Scott queries the DEPT table, the system will SELECT from table SCOTT.DEPT.

If a statement does not use the full name of an object, and the current user does not own an object of that name, the system will prefix the object name with PUBLIC. For example, if user Scott queries the USER_OBJECTS table, and Scott does not own such a table, the system will SELECT from the data dictionary view by way of the PUBLIC.USER_OBJECTS public synonym.

Confirming Privileges Granted

Data Dictionary Table	Description
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
USER_ROLE_PRIVS	Roles accessible by the user
USER_TAB_PRIVS_MADE	Object privileges granted on the user's objects
USER_TAB_PRIVS_RECD	Object privileges granted to the user
USER_COL_PRIVS_MADE	Object privileges granted on the columns of the user's objects
USER_COL_PRIVS_RECD	Object privileges granted to the user on specific columns

Confirming Privileges Granted

If you attempt to perform an unauthorized operation—for example, deleting a row from a table for which you do not have the DELETE privilege—the Oracle8 Server will not permit the operation to take place.

If you receive the Oracle8 Server error message “table or view does not exist,” you have done either of the following:

- Named a table or view that does not exist
- Attempted to perform an operation on a table or view for which you do not have the appropriate privilege

You can access the data dictionary to view the privileges that you have. The table on the slide describes various data dictionary tables.

How to Revoke Object Privileges

- You use the REVOKE statement to revoke privileges granted to other users.
- Privileges granted to others through the WITH GRANT OPTION will also be revoked.

```
REVOKE {privilege [, privilege...]|ALL}  
ON      object  
FROM    {user[, user...]|role|PUBLIC}  
[CASCADE CONSTRAINTS];
```

Revoking Object Privileges

Remove privileges granted to other users by using the REVOKE statement. When you use the REVOKE statement, the privileges that you specify are revoked from the users that you name and from any other users to whom those privileges may have been granted.

In the syntax:

CASCADE	is required to remove any referential integrity constraints made to the
CONSTRAINTS	object by means of the REFERENCES privilege

For more information, see *Oracle8 Server SQL Reference, Release 8.0*, “REVOKE.”



Revoking Object Privileges

As user Alice, revoke the SELECT and INSERT privileges given to user Scott on the DEPT table.

```
SQL> REVOKE  select, insert
      2  ON      dept
      3  FROM    scott;
Revoke succeeded.
```

Revoking Object Privileges (continued)

The example above revokes SELECT and INSERT privileges given to user Scott on the DEPT table.

Note: If a user is granted a privilege WITH GRANT OPTION, that user can also grant the privilege WITH GRANT OPTION, so that a long chain of grantees is possible, but no circular grants are permitted. If the owner revokes a privilege from a user who granted the privilege to other users, the REVOKE cascades to all privileges granted.

For example, if user A grants SELECT privilege on a table to user B including the WITH GRANT OPTION, user B can grant to user C the SELECT privilege WITH GRANT OPTION, and user C can then grant to user D the SELECT privilege. If user A the revokes the privilege from user B, then the privileges granted to users C and D are also revoked.

Summary

CREATE USER	Allows the DBA to create a user
GRANT	Allows the user to give other users privileges to access the user's objects
CREATE ROLE	Allows the DBA to create a collection of privileges
ALTER USER	Allows users to change their password
REVOKE	Removes privileges on an object from users

Summary

DBAs establish initial database security for users by assigning privileges to the users.

- The DBA creates users who must have a password. The DBA is also responsible for establishing the initial system privileges for a user.
- Once the user has created an object, the user can pass along any of the available object privileges to other users or to all users by using the GRANT statement.
- A DBA can create roles by using the CREATE ROLE statement to pass along a collection of system or object privileges to multiple users. Roles make granting and revoking privileges easier to maintain.
- Users can change their password by using the ALTER USER statement.
- You can remove privileges from users by using the REVOKE statement.
- Data dictionary views allow users to view the privileges granted to them and those that are granted on their objects.

Practice Overview

- **Granting other users privileges to your table**
- **Modify another user's table through the privileges granted to you**
- **Creating a synonym**
- **Querying the data dictionary views related to privileges**

Practice Overview

Team up with other students for this exercise of controlling access to database objects.

Practice 14

1. What privilege should a user be given to log in to the Oracle8 Server? Is this privilege a system or object privilege?

2. What privilege should a user be given to create tables?

3. If you create a table, who can pass along privileges to other users on your table?

4. You are the DBA. You are creating many users who require the same system privileges. What would you use to make your job easier?

5. What command do you use to change your password?

6. Grant another user access to your DEPT table. Have the user grant you query access to his or her DEPT table.
7. Query all the rows in your DEPT table.

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

8. Add a new row to your DEPT table. Team 1 should add Education as department number 50. Team 2 should add Administration as department number 50. Make the changes permanent.
9. Create a synonym for the other team's DEPT table.

Practice 14 (continued)

10. Query all the rows in the other team's DEPT table by using your synonym.

Team 1 SELECT statement results.

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	ADMINISTRATION	

Team 2 SELECT statement results.

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	EDUCATION	

11. Query the USER_TABLES data dictionary to see information about the tables that you own.

TABLE_NAME

BONUS
CUSTOMER
DEPARTMENT
DEPT

DUMMY
EMP
EMPLOYEE
ITEM
MY_EMPLOYEE
ORD
PRICE
PRODUCT
SALGRADE
13 rows selected.

Practice 14 (continued)

12. Query the ALL_TABLES data dictionary view to see information about all the tables that you can access. Exclude tables that are owned by you.

TABLE_NAME	OWNER

DEPT	<user2>

13. Revoke the SELECT privilege from the other team.

11

SQL Workshop

Copyright © Oracle Corporation, 1998. All rights reserved. ORACLE®

Schedule:	Timing	Topic
	60-120 minutes	Practice
	60-120 minutes	Total

Workshop Overview

- **Creating tables and sequences**
- **Modifying data in the tables**
- **Modifying a table definition**
- **Creating a view**
- **Writing scripts containing SQL and SQL*Plus commands**
- **Generating a simple report**

Workshop Overview

This workshop has you build a set of database tables for a video application. Once you create the tables, you will insert, update, and delete records in a video store database, and generate a report. The database contains only the essential tables.

Note: If you want to build the tables, you can execute the *buildtab.sql* script in SQL*Plus. If you want to drop the tables, you can execute the *dropvid.sql* script in SQL*Plus. Then you can execute the *buildvid.sql* script in SQL*Plus to create and populate the tables. If you use the *buildvid.sql* to build and populate the tables, start the exercises from p15-6b.

Practice 15

1. Create the tables based on the table instance charts below. Choose the appropriate data types and be sure to add integrity constraints.
 - a. Table name: MEMBER

Column_ Name	MEMBER _ID	LAST_ NAME	FIRST_ NAME	ADDRESS	CITY	PHONE	JOIN_ DATE
Key Type	PK						
Null/ Unique	NN,U	NN					NN
Default Value							System Date
Data Type	Number b. Table name: TITLE	Char	Char	Char	Char	Char	Date
Length	10	25	25	100	30	15	

Column_ Name	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check					G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMENTARY
Data Type	Number	Char	Char	Char	Char	Date
Length	10	60	400	4	20	

Practice 15 (continued)

c. Table name: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
c. Table name: RENTAL	Number	Number	Char
Data Type			
Length	10	10	15

Column Name	BOOK_DATE	MEMBER_ID	COPY_ID	ACT_RET_DATE	EXP_RET_DATE	TITLE_ID
Key Type	PK	PK,FK	PK,FK			PK,FK
Default Value	System Date				2 days	
FK Ref Table		member	copy_id			title_copy
FK Ref Col		member_id				title_id
Data Type	Date	Number	Number	Date	Date	Number
Length		10	10			10

Practice 15 (continued)

e. Table name: RESERVATION

Column_ Name	RES_ DATE	MEMBER_ ID	TITLE_ ID
Key Type	PK	PK,FK	PK,FK
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		member_id	title_id
Data Type	Date	Number	Number
Length		10	10

2. Verify that the tables and constraints were created properly by checking the data dictionary.

```
TABLE_NAME
-----
MEMBER
RENTAL
RESERVATION
TITLE
TITLE_COPY

CONSTRAINT_NAME          C  TABLE_NAME
-----
MEMBER_LAST_NAME_NN      C  MEMBER
MEMBER_JOIN_DATE_NN      C  MEMBER
MEMBER_MEMBER_ID_PK      P  MEMBER
RENTAL_BOOK_DATE_COPY_TITLE_PK  P  RENTAL
RENTAL_MEMBER_ID_FK      R  RENTAL
RENTAL_COPY_ID_TITLE_ID_FK  R  RENTAL
RESERVATION_RESDATE_MEM_TIT_PK  P  RESERVATION
RESERVATION_MEMBER_ID      R  RESERVATION
RESERVATION_TITLE_ID      R  RESERVATION
...
18 rows selected.
```

Practice 15 (continued)

3. Create sequences to uniquely identify each row in the MEMBER table and the TITLE table.
 - a. Member number for the MEMBER table: start with 101; do not allow caching of the values. Name the sequence member_id_seq.
 - b. Title number for the TITLE table: start with 92; no caching. Name the sequence title_id_seq.
 - c. Verify the existence of the sequences in the data dictionary.

SEQUENCE_NAME	INCREMENT_BY	LAST_NUMBER
TITLE_ID_SEQ	1	92
MEMBER_ID_SEQ	1	101

4. Add data to the tables. Create a script for each set of data to add.
 - a. Add movie titles to the TITLE table. Write a script to enter the movie information
Save the script as *p15q4a.sql*. Use the sequences to uniquely identify each title.
Remember that single quotation marks in a character field must be specially handled.

Verify your additions.

```
TITLE
-----
Willie and Christmas Too
Alien Again
The Glob
```

```
My Day Off
Miracles on Ice
Soda Gang
6 rows selected.
```


Practice 15 (continued)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends made a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-95
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-95
The Glob	A meteor crashes near a small American town and unleashed carnivorous goo in this classic.	NR	SCIFI	12-AUG-95
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York	PG	COMEDY	12-JUL-95
Miracles on Ice	A six-year-old has doubts about Santa Claus but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-95
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-95

Practice 15 (continued)

b. Add data to the MEMBER table. Write a script named *p15q4b.sql* to prompt users for the information. Execute the script. Be sure to use the sequence to add the member numbers.

First Name	Last Name	Address	State	Phone	Join Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-90
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-90
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-91
Mark	Lewis	6921 King Way	Lagos	63-559-7777	07-APR-90
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-91
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-91

Practice 15 (continued)

c. Add the following movie copies in the TITLE_COPY table:

Title	Copy Number	Status
Willie and Christmas Too	1	Available
Alien	1	Available
	2	Rented
The Glob	1	Available
My Day Off	1	Available
	2	Available
	3	Rented
Miracles on Ice	1	Available
Soda Gang	1	Available

d. Add the following rentals to the RENTAL table:
 Note: Title number may be different depending on sequence number.

Title	Copy_number	Customer	Date_Rented	Date_return_expected	Date_returned
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

Practice 15 (continued)

5. Create a view named TITLE_AVAIL to show the movie titles and the availability of each copy and its expected return date if rented. Query all rows from the view.

TITLE	COPY_ID	STATUS	EXP_RET_D
-----	-----	-----	-----
Alien Again	1	AVAILABLE	
Alien Again	2	RENTED	05-NOV-97
Miracles on Ice	1	AVAILABLE	
My Day Off	1	AVAILABLE	
My Day Off	2	AVAILABLE	
My Day Off	3	RENTED	06-NOV-97
Soda Gang	1	AVAILABLE	04-NOV-97
The Glob	1	AVAILABLE	
Willie and Christmas Too	1	AVAILABLE	05-NOV-97
9 rows selected.			

6. Make changes to data in the tables.
- Add a new title. The movie is "Interstellar Wars," which is rated PG and classified as a Sci-fi movie. The release date is 07-JUL-77. The description is "Futuristic interstellar action movie. Can the rebels save the humans from the evil Empire?" Be sure to add a title copy record for two copies.
 - Enter two reservations. One reservation is for Carmen Velasquez, who wants to rent "Interstellar Wars." The other is for Mark Lewis, who wants to rent "Soda Gang."

Practice 15 (continued)

- c. Customer Carmen Velasquez rents the movie “Interstellar Wars,” copy 1. Remove her reservation for the movie. Record the information about the rental. Allow the default value for the expected return date to be used. Verify that the rental was recorded by using the view you created.

TITLE	COPY_ID	STATUS	EXP_RET_D
-----	-----	-----	-----
Alien Again	1	AVAILABLE	
Alien Again	2	RENTED	05-NOV-97
Interstellar Wars	1	RENTED	08-NOV-97
Interstellar Wars	2	AVAILABLE	
Miracles on Ice	1	AVAILABLE	
My Day Off	1	AVAILABLE	
My Day Off	2	AVAILABLE	
My Day Off	3	RENTED	06-NOV-97
Soda Gang	1	AVAILABLE	04-NOV-97
The Glob	1	AVAILABLE	
Willie and Christmas Too	1	AVAILABLE	05-NOV-97
9 rows selected.			

7. Make a modification to one of the tables.

a. Add a PRICE column to the TITLE table to record the purchase price of the video. The column should have a total length of eight digits and two decimal places. Verify your modifications.

Name	Null?	Type
-----	-----	-----
TITLE_ID	NOT NULL	NUMBER(10)
TITLE	NOT NULL	VARCHAR2(60)
DESCRIPTION	NOT NULL	VARCHAR2(400)
RATING		VARCHAR2(4)
CATEGORY		VARCHAR2(20)
RELEASE_DATE		DATE
PRICE		NUMBER(8,2)

8. Create a report titled Customer History Report. This report will contain each customer’s history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the script in a file named *p15q8.sql*.

Practice 15 (continued)

- b. Create a script named *p15q7b.sql* to update each video with a price according to the following list. Note: Have the title id numbers available for this exercise.

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	98
Soda Gang	35
Interstellar Wars	29

c. Ensure that in the future all titles will contain a price value. Verify the constraint.

```

CONSTRAINT_NAME      C  SEARCH_CONDITIONS
-----
TITLE_PRICE_NN       C  PRICE IS NOT NULL

```

8. Create a report titles Customer History Report. This report will contain each customer's history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the script in a file name *p15q8.sql*.

```

MEMBER      TITLE      BOOK_DATE  DURATION
-----
LaDoris Ngao  The Glob      04-NOV-97
Molly Urguhart  Miracles on Ice  02-NOV-97      2
Carmen Velasquez  Willie and Christmas  03-NOV-97      1
Too
Willie and Christmas  03-NOV-97      1
Too
Alien Again      05-NOV-97

```

11

Declaring Variables

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	45 minutes	Lecture
	30 minutes	Practice
	75 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Recognize the basic PL/SQL block and its sections**
- **Describe the significance of variables in PL/SQL**
- **Distinguish between PL/SQL and non-PL/SQL variables**
- **Declare PL/SQL variables**
- **Execute a PL/SQL block**

11-112

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Lesson Aim

This lesson presents the basic rules and structure for writing and executing PL/SQL blocks of code. It also shows you how to declare variables and assign datatypes to them.

PL/SQL Block Structure

- **DECLARE – Optional**
 - Variables, cursors, user-defined exceptions
- **BEGIN – Mandatory**
 - SQL statements
 - PL/SQL control statements
- **EXCEPTION – Optional**
 - Actions to perform when errors occur
- **END; – Mandatory**

```
DECLARE
  ○ ○ ○
BEGIN
  ○ ○ ○
EXCEPTION
  ○ ○ ○
END;
```

11-113

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

PL/SQL is a block-structured language, meaning that programs can be divided into logical blocks. A PL/SQL block is comprised of up to three sections: declarative (optional), executable (required), and exception handling (optional). Only BEGIN and END keywords are required. You can declare variables locally to the block that uses them. Error conditions (known as exceptions) can be handled specifically within the block to which they apply. You can store and change values within a PL/SQL block by declaring and referencing variables and other identifiers.

The following table describes the three block sections.

Section	Description	Inclusion
Declarative	Contains all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and declarative sections.	Optional
Executable	Contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.	Mandatory
Exception handling	Specifies the actions to perform when errors and abnormal conditions arise in the executable section	Optional

PL/SQL Block Structure

```
DECLARE
    v_variable  VARCHAR2(5);
BEGIN
    SELECT      column_name
    INTO        v_variable
    FROM        table_name
EXCEPTION
    WHEN exception_name THEN
        ...
END;
```

```
DECLARE
  ○ ○ ○
BEGIN
  ○ ○ ○
EXCEPTION
  ○ ○ ○
END;
```

11-114

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Executing Statements and PL/SQL Blocks from SQL*Plus

- Place a semicolon (;) at the end of a SQL statement or PL/SQL control statement.
- Place a forward slash (/) to run the anonymous PL/SQL block in the SQL buffer. When the block is executed successfully, without unhandled errors or compile errors, the message output should be as follows.
- Place a period (.) to close a SQL buffer. A PL/SQL block is treated as one continuous statement in the buffer, and PL/SQL provides the block to be successfully completed.

Note: In PL/SQL, an error is called an *exception*.

Section keywords DECLARE, BEGIN, and EXCEPTION are not followed by semicolons. However, END and all other PL/SQL statements do require a semicolon to terminate the statement. You can string statements together on the same line. However, this method is not recommended for clarity or editing.

Class Management Note

Inform the class that SQL*Plus is used throughout this course for the execution of demonstration and lesson practices.

Block Types

Anonymous

```
[DECLARE]

BEGIN
    --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS

BEGIN
    --statements

[EXCEPTION]

END;
```

Function

```
FUNCTION name
RETURN datatype
IS

BEGIN
    --statements
    RETURN value;
[EXCEPTION]

END;
```

Every unit of PL/SQL comprises one or more blocks. These blocks can be entirely separate or nested one within another. The basic units (procedures and functions, also known as subprograms, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested subblocks. Therefore one block can represent a small part of another block, which in turn can be part of the whole unit of code. Of the two types of PL/SQL constructs available, anonymous blocks and subprograms, only anonymous blocks are covered in this course.

Anonymous Blocks

Anonymous blocks are unnamed blocks. They are declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at runtime. You can embed an anonymous block within a precompiler program and within SQL*Plus or Server Manager. Triggers in Developer/2000 components consist of such blocks.

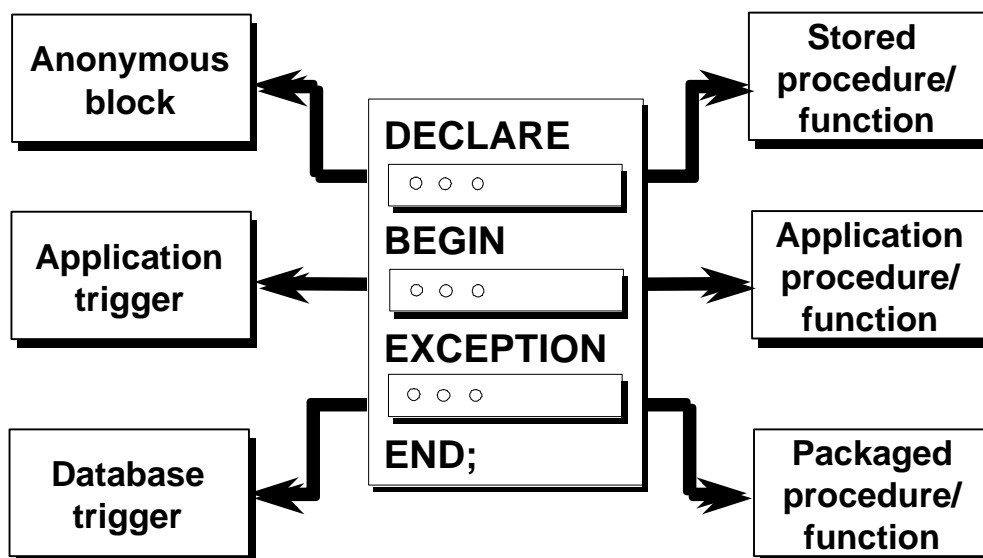
Subprograms

Subprograms are named PL/SQL blocks that can take parameters and can be invoked. You can declare them either as procedures or as functions. Generally you use a procedure to perform an action and a function to compute a value.

You can store subprograms at the server or application level. Using Developer/2000 components (Forms, Reports, and Graphics) you can declare procedures and functions as part of the application (a form or report), and call them from other procedures, functions, and triggers (see next page) within the same application whenever necessary.

Note: A function is similar to a procedure, except that a function *must* return a value. Procedures and functions are covered in the PL/SQL Program Units course.

Program Constructs



11-116

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

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

Program Construct	Description	Availability
Anonymous Block	Unnamed PL/SQL block that is embedded within an application or is issued interactively	All PL/SQL environments
Stored Procedure or Function	Named PL/SQL block stored within the Oracle8 Server that can accept parameters and can be invoked repeatedly by name	Oracle8 Server
Application Procedure or Function	Named PL/SQL block stored within a Developer/2000 application or shared library that can accept parameters and can be invoked repeatedly by name	Developer/2000 components—for example, Forms
Package	Named PL/SQL module that groups together related procedures, functions, and identifiers	Oracle8 Server and Developer/2000 components—for example, Forms
Database Trigger	PL/SQL block that is associated with a database table and is fired automatically when triggered by DML statements	Oracle8 Server
Application Trigger	PL/SQL block that is associated with an application event and is fired automatically	Developer/2000 components—for example, Forms

Use of Variables

Use variables for

- **Temporary storage of data.**
- **Manipulation of stored values.**
- **Reusability.**
- **Ease of maintenance.**

With PL/SQL you can declare variables then use them in SQL and procedural statements anywhere an expression can be used.

- Temporary storage of data

Data can be temporarily stored in one or more variables for use when validating data input for processing later in data flow process.

- Manipulation of stored values

Variables can be used for calculations and other data manipulations without accessing the database.

- Reusability

Once declared, variables can be used repeatedly within an application simply by referencing them in other statements, including other declarative statements.

- Ease of maintenance

When using %TYPE and %ROWTYPE (see Lesson 20 for more information on %ROWTYPE) you declare variables, basing the declarations on the definitions of database columns. PL/SQL variables or cursor variables previously declared within the current scope may also use the %TYPE and %ROWTYPE attributes as datatype specifiers. If an underlying definition changes, the variable declaration changes accordingly at runtime. This provides data independence, reduces maintenance costs, and allows programs to adapt as the database changes to meet new business needs.

Handling Variables in PL/SQL

- **Declare and initialize variables within the declaration section.**
- **Assign new values to variables within the executable section.**
- **Pass values into PL/SQL blocks through parameters.**
- **View results through output variables.**

- Declare and initialize variables within 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 datatype, and name the storage location so that you can reference it. Declarations can also assign an initial value and impose the NOT NULL constraint.

- Assign new values to variables within the executable section.
 - The existing value of the variable is replaced with a new one.
 - Forward references are not allowed. You must declare a variable before referencing it in other statements, including other declarative statements.

- Pass values into PL/SQL subprograms through parameters.

There are three parameter modes, IN (the default), OUT, and IN OUT. You use the IN parameter to pass values to the subprogram being called. You use the OUT parameter to return values to the caller of a subprogram. And you use the IN OUT parameter to pass initial values to the subprogram being called and to return updated values to the caller. IN and OUT subprogram parameters are covered in the PL/SQL Program Units course.

- View the results from a PL/SQL block through output variables.

You can use reference variables for input or output in SQL data manipulation statements.

Types of Variables

- **PL/SQL variables**
 - **Scalar**
 - **Composite**
 - **Reference**
 - **LOB (large objects)**
- **Non-PL/SQL variables**
 - **Bind and host variables**

11-119

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

All PL/SQL variables have a datatype, which specifies a storage format, constraints, and valid range of values. PL/SQL supports four datatype categories—scalar, composite, reference, and LOB (large object)—that you can use for declaring variables, constants, and pointers.

- Scalar datatypes hold a single value. The main datatypes are those that correspond to column types in Oracle8 Server tables; PL/SQL also supports Boolean variables.
- Composite datatypes such as records allow groups of fields to be defined and manipulated in PL/SQL blocks. Composite datatypes are only briefly mentioned in this course.
- Reference datatypes hold values, called *pointers*, that designate other program items. Reference datatypes are not covered in this course.
- LOB datatypes hold values, called *locators*, that specify the location of large objects (graphic images for example) that are stored out of line. LOB datatypes are only briefly mentioned in this course.

Non-PL/SQL variables include host language variables declared in precompiler programs, screen fields in Forms applications, and SQL*Plus bind variables.

Substitution variables in SQL*Plus allow portions of command syntax to be stored and then edited into the command before it is run. These are true host variables in that you can use them to pass runtime values, number or character, into or out of a PL/SQL block. You can then reference them within a PL/SQL block as host variables with a preceding colon.

For more information on LOBs, see
Guide and Reference, Release 8, “Fundamentals.”

PL/SQL User's



Types of Variables

25-OCT-99

TRUE

256120.08

"Four score and seven years ago
our fathers brought forth upon
this continent, a new nation,
conceived in LIBERTY, and dedicated
to the proposition that all men
are created equal."

Atlanta

11-120

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Examples of some of the different variable datatypes in the above illustration are as follows.

- TRUE represents a Boolean value.
- 25-OCT-99 represents a DATE.
- The photograph represents a BLOB.
- The text of a speech represents a LONG RAW.
- 256120.08 represents a NUMBER datatype with precision.
- The movie represents a BFILE.
- The city name represents a VARCHAR2.

Declaring PL/SQL Variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]  
[ := | DEFAULT expr];
```

Examples

```
Declare  
  v_hiredate      DATE;  
  v_deptno        NUMBER(2) NOT NULL := 10;  
  v_location      VARCHAR2(13) := 'Atlanta';  
  c_comm          CONSTANT NUMBER := 1400;
```

11-121

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

You need to declare all PL/SQL identifiers within the declaration section before referencing them within the PL/SQL block. You have the option to assign an initial value. 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, you must be sure to declare them 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.

datatype is a scalar, composite, reference, or LOB datatype (this course covers only scalar and composite datatypes).

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, another variable, or an expression involving operators and functions.

Declaring PL/SQL Variables

Guidelines

- **Follow naming conventions.**
- **Initialize variables designated as NOT NULL.**
- **Initialize identifiers by using the assignment operator (:=) or by using the DEFAULT reserved word.**
- **Declare at most one identifier per line.**

11-122

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Guidelines

The assigned expression can be a literal, another variable, or an expression involving operators and functions.

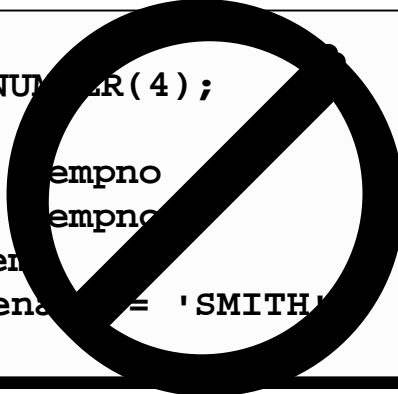
- Name the identifier according to the same rules used for SQL objects.
- You can use naming conventions—for example, v_name to represent a variable and c_name to represent a constant variable.
- Initialize the variable to an expression with the assignment operator (:=) or, equivalently, with the DEFAULT reserved word. If you do not assign an initial value, the new variable contains NULL by default until you assign it later.
- If you use the NOT NULL constraint, you must assign a value.
- By declaring only one identifier per line code is more easily read and maintained.
- In constant declarations, the keyword CONSTANT must precede the type specifier. The following declaration names a constant of NUMBER sub-type REAL and assigns the value of 50000 to the constant. A constant must be initialized in its declaration; otherwise you get a compilation error when the declaration is elaborated (compiled).

```
v_sal      CONSTANT REAL := 50000.00;
```

Naming Rules

- Two variables can have the same name, provided they are in different blocks.
- The variable name (identifier) should not be the same as the name of table columns used in the block.

```
DECLARE
    empno NUMBER(4);
BEGIN
    SELECT empno
    INTO empno
    FROM emp
    WHERE empno = 'SMITH';
END;
```



11-123

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Two objects can have the same name, provided that they are defined in different blocks. Where they coexist, only the object declared in the current block can be used.

You should not choose the same name (identifier) for a variable as the name of table columns used in the block. If PL/SQL variables occur in SQL statements and have the same name as a column, the Oracle8 Server assumes that it is the column that is being referenced. Although the example code in the slide works; code written using the same name for a database table and variable name is not easy to read nor is it easy to maintain.

Consider adopting a naming convention for various objects such as the following example. Using *v_* as a prefix representing *variable*, and *g_* representing *global* variable, avoids naming conflicts with database objects.

```
DECLARE
```

Note: Identifiers must not be longer than 30 characters. The first character must be a letter; the remaining characters may be letters, numbers, or special symbols.

```
    v_hiredate
```

```
    date
```

```
    g_deptno
```

```
    number(2) NOT NULL := 10;
```

```
BEGIN
```

```
    ...
```

Assigning Values to Variables

Syntax

```
identifier := expr;
```

Examples

Set a predefined hiredate for new employees.

```
v_hiredate := '31-DEC-1998';
```

Set the employee name to “Maduro.”

```
v_ename := 'Maduro';
```

To assign or reassign a value to a variable, you write a PL/SQL assignment statement. You must explicitly name the variable to receive the new value to the left of the assignment operator (:=").

In the syntax,

identifier is the name of the scalar variable.

expr can be a variable, literal, or function call, but *not* a database column.

The variable value assignment examples are defined as follows:

- Set the maximum salary identifier V_MAX_SAL to the value of current salary identifier V_SAL.
- Store the name “Maduro” in the v_ename identifier.

Another way to assign values to variables is to select or fetch database values into it. In the following example, you have Oracle compute a 10% bonus when you select the salary of an employee.

```
SELECT sal * 0.10  
INTO bonus  
FROM emp  
WHERE empno = :empno;
```

Then you can use the variable *bonus* in another computation or insert its value into a database table.

Note: To assign a value into a variable from the database, use a SELECT or FETCH statement

Variable Initialization and Keywords

Using

- **:= Assignment Operator**
- **DEFAULT**
- **NOT NULL**

11-125

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Variables are initialized every time a block or subprogram is entered. By default, variables are initialized to NULL. Unless you expressly initialize a variable, its value is undefined.

- Use the assignment (:=) operator for variables that have no typical value.

```
v_hiredate := to_date('15-SEP-99', 'DD-MON-YYYY');
```

Because the default date format set within the Oracle8 Server can differ from database to database, you may want to assign date values in a generic manner, as in the previous example.



DEFAULT—You can use the keyword DEFAULT instead of the assignment operator to initialize variables. Use DEFAULT for variables that have a typical value.

- **NOT NULL**—Impose the NOT NULL constraint when the variable must contain a value.

```
g_mgr NUMBER(4) DEFAULT 7839;
```

You cannot assign nulls to a variable defined as NOT NULL. The NOT NULL constraint must be followed by an initialization clause.

Note: String literals must be enclosed in single quotation marks—for example, 'Hello, world'. If there is a single quotation mark in the string, write a single quotation mark twice—for example, 'Account wasn't found'.

```
v_location VARCHAR2(13) NOT NULL := 'CHICAGO';
```

Scalar Datatypes

- Hold a single value.
- Have no internal components.

25-OCT-00

256120.08

"Four score and seven years
ago our fathers brought
forth upon this continent, a
new nation, conceived in
LIBERTY, and dedicated to
the proposition that all men
are created equal."

TRUE

Atlanta

11-126

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

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

For more information and the complete list of scalar datatypes, see
Guide and Reference, Release 8, "Fundamentals."

PL/SQL User's



Base Scalar Datatypes

- **VARCHAR2**(*maximum_length*)
- **NUMBER** [(*precision*, *scale*)]
- **DATE**
- **CHAR** [(*maximum_length*)]
- **LONG**
- **LONG RAW**
- **BOOLEAN**
- **BINARY_INTEGER**

11-127

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Datatype	Description
VARCHAR2 (<i>maximum_length</i>)	Base type for variable-length character data up to 32767 bytes. There is no default size for VARCHAR2 variables and constants.
NUMBER [(<i>precision</i> , <i>scale</i>)]	Base type for fixed and floating point numbers.
DATE	Base type for dates and times. DATE values include the time of day in seconds since midnight. The range for dates is between 4712 BC and 9999 AD.
CHAR [(<i>maximum_length</i>)]	Base type for fixed length character data up to 32767 bytes. If you do not specify a <i>maximum_length</i> , the default length is set to 1.
LONG	Base type for variable-length character data up to 32760 bytes. The maximum width of a LONG database column is 2147483647 bytes.
LONG RAW	Base type for binary data and byte strings up to 32760 bytes. LONG RAW data is not interpreted by PL/SQL.
BOOLEAN	Base type that stores one of three possible values used for logical calculations: TRUE, FALSE, or NULL.
BINARY_INTEGER	Base type for integers between -2147483647 and 2147483647.

Note: The LONG datatype is similar to VARCHAR2, except that the maximum length of a LONG value is 32,760 bytes. Therefore values longer than 32,760 bytes cannot be stored in a LONG database column into a LONG PL/SQL variable.

Scalar Variable Declarations

Examples

```
v_job          VARCHAR2(9);  
v_count        BINARY_INTEGER := 0;  
v_total_sal    NUMBER(9,2) := 0;  
v_orderdate    DATE := SYSDATE + 7;  
c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;  
v_valid        BOOLEAN NOT NULL := TRUE;
```

Declaring Scalar Variables

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

- Declared variable to store an employee job title.
- Declared variable to count the iterations of a loop and initialize the variable to 0.
- Declared variable to accumulate the total salary for a department and initialize the variable to 0.
- Declared variable to store the ship date of an order and initialize the variable to one week from today.
- Declared a constant variable for the tax rate, which never changes throughout the PL/SQL block.
- Declared flag to indicate whether a piece of data is valid or invalid and initialize the variable to TRUE.

The %TYPE Attribute

- **Declare a variable according to**
 - **A database column definition.**
 - **Another previously declared variable.**
- **Prefix %TYPE with**
 - **The database table and column.**
 - **The previously declared variable name.**

When you declare PL/SQL variables to hold column values, you must ensure that the variable is of the correct datatype and precision. If it is not, a PL/SQL error will occur during execution.

Rather than hard coding the datatype and precision of a variable, you can use the %TYPE attribute to declare a variable according to another previously declared variable or database column. The %TYPE attribute is most often used when the value stored in the variable will be derived from a table in the database or if the variable is destined to be written out to it. To use the attribute in place of the datatype required in the variable declaration, prefix it with the database table and column names. If referring to a previously declared variable, prefix the variable name to the attribute.

PL/SQL determines the datatype and size of the variable when the block is compiled, so it is always compatible with the column used to populate it. This is a definite advantage for writing and maintaining code, because there is no need to be concerned with column datatype changes made at the database level. You can also declare a variable according to another previously declared variable by prefixing the variable name to the attribute.

Class Management Note

The %TYPE attribute has some overhead, in that a SELECT statement is issued against the database to obtain the datatype. If the PL/SQL code is in a client tool, the SELECT must be executed each time the block is executed. If the PL/SQL code is a stored procedure, the column or definition is stored as part of the P-code. However, if the table definition changes, a recompile is forced.

Declaring Variables with the %TYPE Attribute

Examples

```
...  
  v_ename          emp.ename%TYPE;  
  v_balance        NUMBER(7,2);  
  v_min_balance    v_balance%TYPE := 10;  
...
```

11-130

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Declare variables to store the name an employee.

```
...  
v_ename          emp.ename%TYPE;  
...
```

Declare variables to store the balance of a bank account, as well as the minimum balance, which starts out as 10.

```
...  
v_balance        NUMBER(7,2);  
v_min_balance    v_balance%TYPE := 10;
```

A NOT NULL column constraint does not apply to variables declared using %TYPE. Therefore, if you declare a variable using the %TYPE attribute using a database column defined as NOT NULL, you can assign the NULL value to the variable.



Declaring BOOLEAN Variables

- Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable.
- The variables are connected by the logical operators AND, OR, and NOT.
- The variables always yield TRUE, FALSE, or NULL.
- Arithmetic, character, and date expressions may be used to return a Boolean value.

11-131

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

With PL/SQL you can compare variables and in both SQL and procedural statements. These comparisons, called *Boolean expressions*, consist of simple or complex expressions separated by relational operators. In a SQL statement, you can use Boolean expressions to specify the rows in a table that are affected by the statement. In a procedural statement, Boolean expressions are the basis for conditional control.

NULL stands for a missing, inapplicable, or unknown value.

Examples

The following expression yields TRUE

```
v_sal1 := 50000;  
v_sal2 := 60000;
```

Declare and initialize a Boolean variable.

```
v_sal1 < v_sal2
```

```
v_comm_sal BOOLEAN := (v_sal1 < v_sal2);
```

Composite Datatypes

Types

- **PL/SQL TABLES**
- **PL/SQL RECORDS**

11-132

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

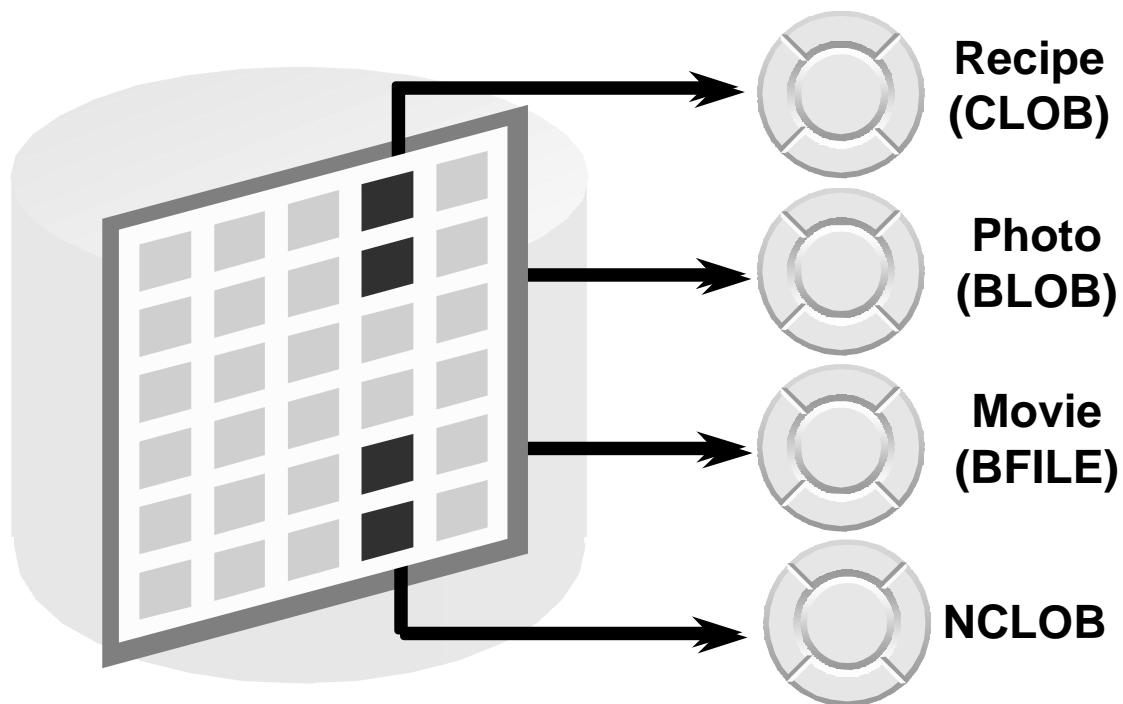
Composite datatypes (also known as *collections*) are TABLE, RECORD, Nested TABLE, and VARRAY. You use the RECORD datatype to treat related but dissimilar data as a logical unit. You use the TABLE datatype to reference and manipulate collections of data as a whole object. Both RECORD and TABLE datatypes are covered in detail in Lesson 20. The Nested TABLE and VARRAY datatypes are not covered in this course.

For more information, see
Guide and Reference, Release 8, “Collections and Records.”

PL/SQL User's



LOB Datatype Variables



11-133

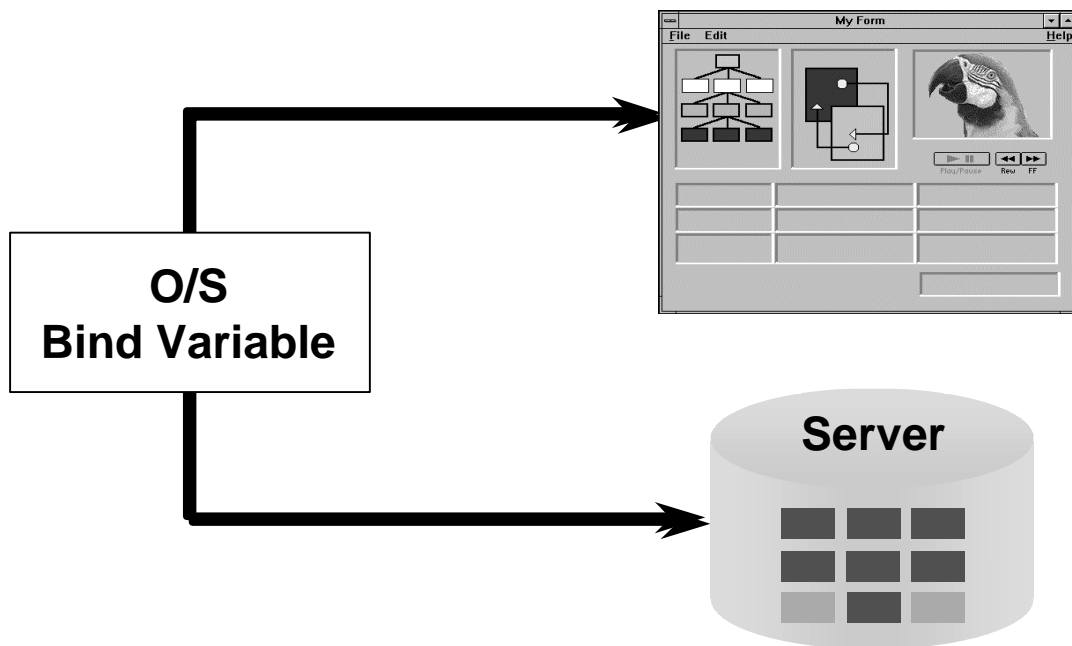
Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

With the LOB (large object) Oracle8 Server datatypes you can store blocks of unstructured data (such as text, graphic images, video clips, and sound wave forms) up to 4 gigabytes in size. LOB datatypes allow efficient, random, piece-wise access to the data and can be attributes of an object type. LOBs also support random access to data.

- The CLOB (character large object) datatype is used to store large blocks of single-byte character data in the database.
- The BLOB (binary large object) datatype is used to store large binary objects in the database in line (inside the row) or out of line (outside the row).
- The BFILE (binary file) datatype is used to store large binary objects in operating system files outside the database.
- The NCLOB (national language character large object) datatype is used to store large blocks of single-byte or fixed-width multi-byte NCHAR data in the database, in line or out of line.

Bind Variables



11-134

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

A bind variable is a variable that you declare in a host environment, then use to pass runtime values, either number or character, into or out of one or more PL/SQL programs, which can use it like any other variable. You can reference variables declared in the host or calling environment in PL/SQL statements, unless the statement is within a procedure, function, or package. This includes host language variables declared in precompiler programs, screen fields in Developer/2000 Forms applications, and SQL*Plus bind variables.

Creating Bind Variables

In the SQL*Plus environment, to declare a bind variable, you use the command `VARIABLE`. For example, you declare a variable of type `NUMBER` as follows:

Both SQL and SQL*Plus can reference the bind variable, and SQL*Plus can display its value.

```
VARIABLE return_code NUMBER
```

Referencing Non-PL/SQL Variables

Store the annual salary into a SQL*Plus global variable.

```
:g_monthly_sal := v_sal / 12;
```

- **Reference non-PL/SQL variables as host variables.**
- **Prefix the references with a colon (:).**

Assigning Values to Variables

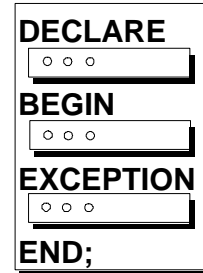
To reference host variables, you must prefix the references with a colon (:) to distinguish them from declared PL/SQL variables.

Examples

```
:host_var1 := g_monthly_sal;  
:global_var1 := 'YES';
```

Summary

- **PL/SQL blocks**
 - **Are composed of the following sections:**
 - Declarative (optional).
 - Executable (required).
 - Exception handling (optional).
 - **Can be an anonymous block, procedure, or function.**



Summary

- **PL/SQL identifiers**
 - **Are defined in the declarative section.**
 - **Can be of scalar, composite, reference, or LOB datatype.**
 - **Can be based on the structure of another variable or database object.**
 - **Can be initialized.**

Practice Overview

- **Determining validity of declarations**
- **Developing a simple PL/SQL block**

Practice Overview

This practice reinforces the basics of PL/SQL learned in this lesson, including data types, legal definitions of identifiers, and validation of expressions. You put all these elements together to create a simple PL/SQL block.

Practice 16

Declare variables.

1. Evaluate each of the following declarations. Determine which of them are *not* legal and explain why.

a. DECLARE

```
v_id          NUMBER(4);
```

b. DECLARE

```
v_x, v_y, v_z    VARCHAR2(10);
```

c. DECLARE

```
v_birthdate      DATE NOT NULL;
```

d. DECLARE

```
v_in_stock       BOOLEAN := 1;
```

e. DECLARE

```
emp_record       emp_record_type;
```

f. DECLARE

```
TYPE name_table_type IS TABLE OF VARCHAR2(20)  
    INDEX BY BINARY_INTEGER;  
dept_name_table  name_table_type;
```

Practice 16 (continued)

2. In each of the following assignments, determine the data type of the resulting expression.

a. `v_days_to_go := v_due_date - SYSDATE;`

b. `v_sender := USER || ':' || TO_CHAR(v_dept_no);`

c. `v_sum := $100,000 + $250,000;`

d. `v_flag := TRUE;`

e. `v_n1 := v_n2 > (2 * v_n3);`

f. `v_value := NULL;`

3. Create an anonymous block to output the phrase “My PL/SQL Block Works” to the screen.

```
G|MESSAGE
-|-----
M| My PL/SQL Block Works
```

Practice 16 (continued)

If you have time, complete the following exercise.

4. Create a block that declares two variables. Assign the value of these PL/SQL variables to SQL*Plus variables and print the results of the PL/SQL variables to the screen. Execute your PL/SQL block. Save your PL/SQL block to a file named *p1q4.sql*.

```
V_CHAR Character (variable length)
```

```
V_NUM Number
```

Assign values to these variables as follows:

```
Variable Value
```

```
-----
```

```
V_CHAR The literal '42 is the answer'
```

```
V_NUM The first two characters from V_CHAR
```

```
G_CHAR
```

```
-----
```

```
42 is the answer
```

```
G_NUM
```

```
-----
```

```
42
```


11

Writing Executable Statements

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	30 minutes	Lecture
	25 minutes	Practice
	55 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Recognize the significance of the executable section**
- **Write statements within the executable section**
- **Describe the rules of nested blocks**
- **Execute and test a PL/SQL block**
- **Use coding conventions**

Aim

In this lesson, you will learn how to write executable code within the PL/SQL block. You will also learn the rules for nesting PL/SQL blocks of code, as well as how to execute and test their PL/SQL code.

Anatomy of a PL/SQL Block

- **DECLARE – Optional**
 - Variables, constants, cursors, user-defined exceptions
- **BEGIN – Mandatory**
 - SQL statements
 - PL/SQL control statements
- **EXCEPTION – Optional**
 - Actions to perform when errors occur
- **END; – Mandatory**

```
DECLARE
  ○ ○ ○
BEGIN
  ○ ○ ○
EXCEPTION
  ○ ○ ○
END;
```

```
DECLARE
    v_variable  VARCHAR2(5)
BEGIN
    SELECT      column_name
              INTO  v_variable
    FROM        table_name

END;
```

PL/SQL Block Structure Review

PL/SQL is a block-structured language, meaning that programs can be divided into logical blocks. A PL/SQL block is comprised of up to three sections: declarative (optional), executable (required), and exception handling (optional). Only BEGIN and END keywords are required. You can declare variables locally to the block that uses them. Error conditions (known as exceptions) can be handled specifically within the block to which they apply. You can store and change values within a PL/SQL block by declaring and referencing variables and other identifiers.

The following table describes the three block sections.

Section	Description	Inclusion
Declarative	Contains all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and declarative sections.	Optional
Executable	Contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.	Mandatory
Exception Handling	Specifies the actions to perform when errors and abnormal conditions arise in the executable section	Optional

PL/SQL Block Syntax and Guidelines

- **Statements can continue over several lines.**
- **Lexical units can be separated by spaces:**
 - **Delimiters**
 - **Identifiers**
 - **Literals**
 - **Comments**

11-146

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Because PL/SQL is an extension of SQL, the general syntax rules that apply to SQL are also applicable to the PL/SQL language.

- Lexical units (for example, identifiers or literals) can be separated by one or more spaces or other delimiters that cannot be confused as being part of the lexical unit. You cannot embed spaces in lexical units except for string literals and comments.
- Statements can be split across lines, but keywords must not be split.

Delimiters

Delimiters are simple or compound symbols that have special meaning to PL/SQL.

Simple Symbols

Compound Symbols

Symbol	Meaning	Symbol	Meaning
+	Addition Operator	<>	Relational Operator
-	Subtraction/Negation Operator	!=	Relational Operator
*	Multiplication Operator		Concatenation Operator
/	Division Operator	--	Single Line Comment Indicator
=	Relational Operator	/*	Beginning Comment Delimiter
@	Remote Access Indicator	*/	Ending Comment Delimiter
;	Statement Terminator	:=	Assignment Operator



PL/SQL Block Syntax and Guidelines

Identifiers

- **Can contain up to 30 characters.**
- **Cannot contain reserved words unless enclosed in double quotation marks.**
- **Must begin with an alphabetic character.**
- **Should not have the same name as a database table column name.**

11-147

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Identifiers

Identifiers are used to name PL/SQL program items and units, which include constants, variables, exceptions, cursors, cursor variables, subprograms, and packages.

- Identifiers can contain up to 30 characters, but they must start with an alphabetic character.
- Do not choose the same name for the identifier as the name of columns in a table used in the block. If PL/SQL identifiers are in the same SQL statements and have the same name as a column, then Oracle assumes that it is the column that is being referenced.
- Reserved words cannot be used as identifiers unless they are enclosed in double quotation marks (for example, "SELECT").
- Reserved words should be written in uppercase to promote readability.

For a complete list of reserved words, see
Guide and Reference, Release 8, "Appendix F."

PL/SQL User's



PL/SQL Block Syntax and Guidelines

Literals

- **Character and date literals must be enclosed in single quotation marks.**

```
v_ename := 'Henderson';
```

- **Numbers can be simple values or scientific notation.**

Literals

A literal is an explicit numeric, character, string, or Boolean value not represented by an identifier.

- Character literals include all the printable characters in the PL/SQL character set: letters, numerals, spaces, and special symbols.
- Numeric literals can be represented by either a simple value (for example, -32.5) or by scientific notation (for example, 2E5, meaning 2x10 to the power of 5 = 200000).

Commenting Code

- Prefix single-line comments with two dashes (- -).
- Place multi-line comments between the symbols /* and */.

Example

```
...  
    v_sal NUMBER (9,2);  
BEGIN  
    /* Compute the annual salary based on the  
       monthly salary input from the user */  
    v_sal := v_sal * 12;  
END; -- This is the end of the transaction
```

11-149

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Comment code to document each phase and to assist with debugging. Comment the PL/SQL code with two dashes (--) if the comment is on a single line, or enclose the comment between the symbols /* and */ if the comment spans several lines. Comments are strictly informational and do not enforce any conditions or behavior on behavioral logic or data. Well-placed comments are extremely valuable for code readability and future code maintenance.

Example

Compute the yearly salary from the monthly salary.

```
...  
    v_sal NUMBER(9,2);  
BEGIN  
    /* Compute the annual salary based on  
       the monthly salary input from the user */  
    v_sal := v_sal*12;  
END; -- This is the end of the transaction
```

SQL Functions in PL/SQL

- **Available:**

- **Single-row number**
- **Single-row character**
- **Datatype conversion**
- **Date**

} **Same as in SQL**

- **Not available:**

- **GREATEST**
- **LEAST**
- **Group functions**

11-150

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Most of the functions available in SQL are also valid in PL/SQL expressions:

- Single-row number functions
- Single-row character functions
- Datatype conversion functions
- Date functions
- Miscellaneous functions

The following functions are not available in procedural statements:

- GREATEST, LEAST, and DECODE.
- Group functions: AVG, MIN, MAX, COUNT, SUM, STDDEV, and VARIANCE. Group functions apply to groups of rows in a table and are therefore available only within SQL statements in a PL/SQL block.

Example

Compute the sum of all numbers stored in the NUMBER_TABLE PL/SQL table. *This example produces a compile error.*

```
v_total      := SUM(number_table);
```

PL/SQL Functions

Examples

- **Build the mailing list for a company.**

```
v_mailing_address := v_name || CHR(10) ||  
                    v_address || CHR(10) || v_state ||  
                    CHR(10) || v_zip;
```

- **Convert the employee name to lowercase.**

```
v_ename          := LOWER(v_ename);
```

PL/SQL provides many powerful functions to help you manipulate data. These built-in functions fall into the following categories:

- Error-reporting
- Number
- Character
- Conversion
- Date
- Miscellaneous

The function examples in the slide are defined as follows:

Build the mailing address for a company.

Convert the name to lowercase.

CHR is the SQL function that converts an ASCII code to its corresponding character; 10 is the code for a line feed.

For more information, see

Guide and Reference, Release 8, “Fundamentals.”

PL/SQL User's



Datatype Conversion

- Convert data to comparable datatypes.
- Mixed datatypes can result in an error and affect performance.
- Conversion functions:
 - TO_CHAR
 - TO_DATE
 - TO_NUMBER

```
BEGIN
    SELECT TO_CHAR(hiredate,
        'MON. DD, YYYY')
    FROM emp;
END;
```

11-152

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

PL/SQL attempts to convert datatypes dynamically if they are mixed within a statement. For example, if you assign a NUMBER value to a CHAR variable, then PL/SQL dynamically translates the number into a character representation, so that it can be stored in the CHAR variable. The reverse situation also applies, providing that the character expression represents a numeric value.

Providing that they are compatible, you can also assign characters to DATE variables, and vice versa.

Within an expression, you should make sure that datatypes are the same. If mixed datatypes occur in an expression, you should use the appropriate conversion function to convert the data.

Syntax

TO_CHAR (value, fmt)

TO_DATE (value, fmt)

where: *value* is a character string, number, or date.

fmt is the format model used to convert value.

The query result in the example in the slide is DEC. 17, 1998.

Datatype Conversion

This statement produces a compile error.

```
v_comment := USER || ':' || SYSDATE;
```

To correct the error, the TO_CHAR conversion function is used.

```
v_comment := USER || ':' || TO_CHAR(SYSDATE);
```

11-153

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The conversion examples in the slide are defined as follows:

- Store a value that is composed of the user name and today's date. *This code causes a syntax error.*
- To correct the error, convert SYSDATE to a character string with the TO_CHAR conversion function.

PL/SQL attempts conversion if possible, but the success depends on the operations being performed. It is good programming practice to explicitly perform datatype conversions, because they can favorably affect performance and remain valid even with a change in software versions.



Class Management Note

This slide has the build feature first displaying a problem and then displaying the problem and the answer. Upon displaying the problem, give the students an opportunity to solve it.

Nested Blocks and Variable Scope

- **Statements can be nested wherever an executable statement is allowed.**
- **A nested block becomes a statement.**
- **An exception section can contain nested blocks.**
- **The scope of an object is the region of the program that can refer to the object.**

Nested Blocks

One of the advantages that PL/SQL has over SQL is the ability to nest statements. You can nest blocks wherever an executable statement is allowed, thus making the nested block a statement. Therefore you can break down the executable part of a block into smaller blocks. The exception section can also contain nested blocks.

Variable Scope

The scope of an object is the region of the program that can refer to the object. You can reference the declared variable within the executable section.

Nested Blocks and Variable Scope

- An identifier is visible in the regions in which you can reference the unqualified identifier:
 - A block can look up to the enclosing block.
 - A block cannot look down to enclosed blocks.

Identifiers

An identifier is visible in the block in which it is declared and in all nested subblocks, procedures, and functions. If the block does not find the identifier declared locally, it looks *up* to the declarative section of the enclosing (or parent) blocks. The block never looks *down* to enclosed (or child) blocks or sideways to sibling blocks.

Scope applies to all declared objects, including variables, cursors, user-defined exceptions, and constants.

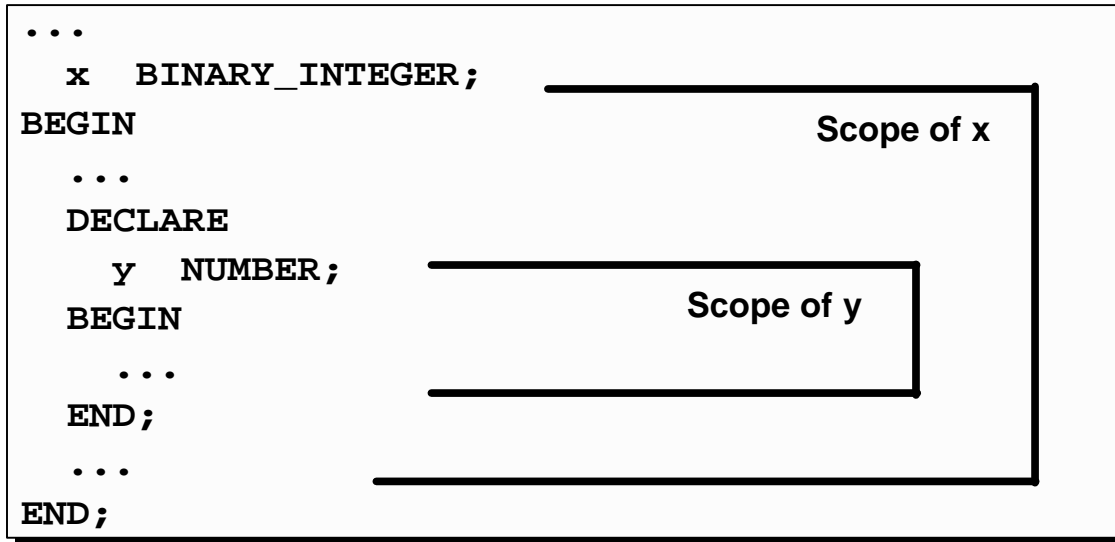
Note: Qualify an identifier by using the block label prefix.

For more information on block labels, see *PL/SQL User's Guide and Reference, Release 8, "Fundamentals."*



Nested Blocks and Variable Scope

Example



11-156

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

In the nested block shown in the slide, the variable named *y* can reference the variable named *x*. Variable *x*, however, cannot reference variable *y*. Had the variable named *y* in the nested block been given the same name as the variable named *x* in the outer block its value is valid only for the duration of the nested block.

Scope

The scope of an identifier is that region of a program unit (block, subprogram, or package) from which you can reference the identifier.

Visibility

An identifier is visible only in the regions from which you can reference the identifier using an unqualified name.

Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations
- Exponential operator (**)



Same as in
SQL

11-157

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Order of Operations

The operations within an expression are done in a particular order depending on their precedence (priority). The following table shows the default order of operations from top to bottom.

Operator	Operation
**, NOT	Exponentiation, logical negation
+, -	Identity, negation
*, /	Multiplication, division
+, =,	Addition, subtraction, concatenation
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparison
AND	Conjunction
OR	Inclusion

Note: It is not necessary to use parentheses with Boolean conjunction, but it does make the text easier to read.

For more information on operators, see

PL/SQL User's

Guide and Reference, Release 8, "Fundamentals."



Operators in PL/SQL

Examples

- **Increment the index for a loop.**

```
v_count      := v_count + 1;
```

- **Set the value of a Boolean flag.**

```
v_equal      := (v_n1 = v_n2);
```

- **Validate an employee number if it contains a value.**

```
v_valid      := (v_empno IS NOT NULL);
```

When working with nulls, you can avoid some common mistakes by keeping in mind the following rules:

- Comparisons involving nulls always yield NULL.
- Applying the logical operator NOT to a null yields NULL.
- In conditional control statements, if the condition yields NULL, its associated sequence of statements is not executed.

Using Bind Variables

To reference a bind variable in PL/SQL, you must prefix its name with a colon (:).

Example

```
:return_code := 0;  
IF credit_check_ok(acct_no) THEN  
    :return_code := 1;  
END IF;
```

Printing Bind Variables

In SQL*Plus you can display the value of the bind variable using the PRINT command.

```
SQL> PRINT return_code
```

```
RETURN_CODE
```

```
-----
```

```
1
```

Programming Guidelines

Make code maintenance easier by

- Documenting code with comments.
- Developing a case convention for the code.
- Developing naming conventions for identifiers and other objects.
- Enhancing readability by indenting.

11-160

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Follow these programming guidelines to produce clear code and reduce maintenance when developing a PL/SQL block.

Code Conventions

The following table gives guidelines for writing code in uppercase or lowercase to help you to distinguish keywords from named objects.

Category	Case Convention	Examples
SQL statements	Uppercase	SELECT, INSERT
PL/SQL keywords	Uppercase	DECLARE, BEGIN, IF
Datatypes	Uppercase	VARCHAR2, BOOLEAN
Identifiers and parameters	Lowercase	v_sal, emp_cursor, g_sal, p_empno
Database tables and columns	Lowercase	emp, orderdate, deptno

Code Naming Conventions

Avoid ambiguity:

- The names of local variables and formal parameters take precedence over the names of database tables.
- The names of columns take precedence over the names of local variables.

The following table shows a set of prefixes and suffixes to distinguish identifiers from other identifiers, from database objects, and from other named objects.

Identifier	Naming Convention	Example
Variable	<i>v_name</i>	v_sal
Constant	<i>c_name</i>	c_company_name
Cursor	<i>name_cursere</i>	emp_cursor
Exception	<i>e_name</i>	e_too_many
Table Type	<i>name_table_type</i>	amount_table_type
Table	<i>name_table</i>	order_total_table
Record Type	<i>name_record_type</i>	emp_record_type
Record	<i>name_record</i>	customer_record
SQL*Plus substitution parameter	<i>p_name</i>	p_sal
SQL*Plus global variable	<i>g_name</i>	g_year_sal

Indenting Code

For clarity, indent each level of code.

Example

```
BEGIN
  IF x=0 THEN
    y=1;
  END IF;
END;
```

```
DECLARE
  v_deptno      NUMBER(2);
  v_location    VARCHAR2(13);
BEGIN
  SELECT deptno,
         location
  INTO   v_deptno,
         v_location
  FROM   dept
  WHERE  dname = 'SALES';
  ...
END;
```

11-162

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

For clarity, and to enhance readability, indent each level of code. To show structure, you can divide lines using carriage returns and indent lines using spaces or tabs. Compare the following IF statements for readability.

```
IF x>y THEN max:=x;ELSE max:=y;END IF;
```

```
IF x > y THEN
  max := x;
ELSE
  max := y;
END IF;
```

Class Management Note

It is not important that students adopt the particular conventions suggested here for case and names, only that they adopt some convention. One of the chief benefits of adopting a standard is avoiding the ambiguous reference that may arise between database column names and PL/SQL variable names.

Determine Variable Scope

Class Exercise

```
...  
DECLARE  
V_SAL          NUMBER(7,2) := 60000;  
V_COMM         NUMBER(7,2) := V_SAL / .20;  
V_MESSAGE      VARCHAR2(255) := ' eligible for commission';  
BEGIN ...  
  
    DECLARE  
        V_SAL          NUMBER(7,2) := 50000;  
        V_COMM         NUMBER(7,2) := 0;  
        V_TOTAL_COMP   NUMBER(7,2) := V_SAL + V_COMM;  
    BEGIN ...  
        V_MESSAGE := 'CLERK not' || V_MESSAGE;  
    END;  
  
    V_MESSAGE := 'SALESMAN' || V_MESSAGE;  
END;
```

11-163

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Class Exercise

Evaluate the PL/SQL block on the slide. Determine each of the following values according to the rules of scoping.

1. The value of V_MESSAGE in the subblock.
2. The value of V_TOTAL_COMP in the main block.
3. The value of V_COMM in the subblock.
4. The value of V_COMM in the main block.
5. The value of V_MESSAGE in the main block.

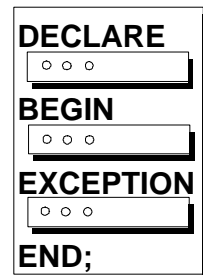
Class Management Note

Answers:

1. V_MESSAGE = 'CLERK not eligible for commission.'
2. V_TOTAL_COMP is illegal since it is not visible outside the subblock.
3. V_COMM = 50000.
4. V_COMM = 12000.
5. V_MESSAGE = 'SALESMAN eligible for commission.'

Summary

- **PL/SQL blocks are composed of the following sections:**
 - **Declarative (optional)**
 - **Executable (required)**
 - **Exception handling (optional)**
- **PL/SQL block structure:**
 - **Nesting blocks and scoping rules**
 - **Executing blocks**



Summary

- **PL/SQL programming:**
 - **Symbols**
 - **Functions**
 - **Datatype conversions**
 - **Operators**
 - **Bind variables**
 - **Conventions and guidelines**

```
DECLARE
  ○ ○ ○
BEGIN
  ○ ○ ○
EXCEPTION
  ○ ○ ○
END;
```

Practice Overview

- **Reviewing scoping and nesting rules**
- **Developing and testing PL/SQL blocks**

11-166

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Practice Overview

This practice reinforces the basics of PL/SQL presented in the lesson, including the rules for nesting PL/SQL blocks of code as well as how to execute and test their PL/SQL code.

Practice 17

PL/SQL Block

```
DECLARE
    v_weight  NUMBER(3) := 600;
    v_message VARCHAR2(255) := 'Product 10012';
BEGIN

    SUB-BLOCK

    DECLARE
        v_weight      NUMBER(3) := 1;
        v_message     VARCHAR2(255) := 'Product 11001';
        v_new_locn    VARCHAR2(50) := 'Europe';
    BEGIN
        v_weight := v_weight + 1;
        v_new_locn := 'Western ' || v_new_locn;
    END;

    v_weight := v_weight + 1;
    v_message := v_message || ' is in stock';
    v_new_locn := 'Western ' || v_new_locn;

END;
```

Practice 17 (continued)

1. Evaluate the PL/SQL block on the previous page and determine each of the following values according to the rules of scoping.

a. The value of V_WEIGHT in the subblock is

b. The value of V_NEW_LOCN in the subblock is

c. The value of V_WEIGHT in the main block is

d. The value of V_MESSAGE in the main block is

e. The value of V_NEW_LOCN in the main block is

Practice 17 (continued)

Scope Example

DECLARE

v_customer VARCHAR2(50) := 'Womansport';

v_credit_rating VARCHAR2(50) := 'EXCELLENT';

BEGIN

DECLARE

v_customer NUMBER(7) := 201;

v_name VARCHAR2(25) := 'Unisports';

BEGIN

v_customer v_name v_credit_rating

END;

v_customer v_name v_credit_rating

END;

Practice 17 (continued)

2. Suppose you embed a subblock within a block, as shown on the previous page. You declare two variables, V_CUSTOMER and V_CREDIT_RATING, in the main block. You also declare two variables, V_CUSTOMER and V_NAME, in the subblock. Determine the values for each of the following cases.

a. The value of V_CUSTOMER in the subblock is

b. The value of V_NAME in the subblock is

c. The value of V_CREDIT_RATING in the subblock is

d. The value of V_CUSTOMER in the main block is

e. The value of V_NAME in the main block is

f. The value of V_CREDIT_RATING in the main block is

Practice 17 (continued)

3. Create and execute a PL/SQL block that accepts two numbers through SQL*Plus variables. The first number should be divided by the second number and have the second number added to the result. The result should be written to a PL/SQL variable and printed to the screen.

```
Please enter the first number: 2
Please enter the second number: 4

PL/SQL procedure successfully completed.

V_RESULT
-----
      4.5
```

4. Build a PL/SQL block that computes the total compensation for one year. The annual salary and the annual bonus percentage are passed to the PL/SQL block through SQL*Plus substitution variables and the bonus needs to be converted from a whole number to a decimal (for example, 15 to .15). If the salary is null, set it to zero before computing the total compensation. Execute the PL/SQL block. Reminder: Use the NVL function to handle null values.

Note: To test the NVL function type NULL at the prompt; pressing [Return] results in a missing expression error.

```
Please enter the salary amount: 50000
Please enter the bonus percentage: 10

PL/SQL procedure successfully completed.

G_TOTAL
-----
55000
```


11

Interacting with the Oracle8 Server

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	45 minutes	Lecture
	40 minutes	Practice
	85 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Write a successful SELECT statement in PL/SQL**
- **Declare the datatype and size of a PL/SQL variable dynamically**
- **Write DML statements in PL/SQL**
- **Control transactions in PL/SQL**
- **Determine the outcome of SQL DML statements**

11-174

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Aim

In this lesson, you will learn to embed standard SQL SELECT, INSERT, UPDATE, and DELETE statements in PL/SQL blocks. You will also learn how to control transactions and determine the outcome of SQL DML statements in PL/SQL.

SQL Statements in PL/SQL

- **Extract a row of data from the database by using the SELECT command. Only a single set of values can be returned.**
- **Make changes to rows in the database by using DML commands.**
- **Control a transaction with the COMMIT, ROLLBACK, or SAVEPOINT command.**
- **Determine DML outcome with implicit cursors.**

11-175

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Overview

When you need to extract information from or apply changes to the database, you must use SQL. PL/SQL supports full data manipulation language and transaction control commands within SQL. You can use SELECT statements to populate variables with values queried from a row in a table. Your DML (data manipulation) commands can process multiple rows.

Comparing SQL and PL/SQL Statement Types

- A PL/SQL block is not a transaction unit. Commits, savepoints, and rollbacks are independent of blocks, but you can issue these commands within a block.
- PL/SQL does not support data definition language (DDL), such as CREATE TABLE, ALTER TABLE, or DROP TABLE.
- PL/SQL does not support data control language (DCL), such as GRANT or REVOKE.

For more information about the DBMS_SQL package, see
Application Developer's Guide, Release 8.

Oracle8 Server



Class Management Note

You can mention that with the DBMS_SQL package, you can issue DDL and DCL statements.

SELECT Statements in PL/SQL

Retrieve data from the database with
SELECT.

Syntax

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

11-176

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Retrieving Data Using PL/SQL

Use the SELECT statement to retrieve data from the database.

In the syntax,

select_list is a list of at least one column, and can include SQL expressions, row functions, or group functions.

variable_name is the scalar variable to hold the retrieved value or.

record_name is the PL/SQL RECORD to hold the retrieved values.

table specifies the database table name.

condition is composed of column names, expressions, constants, and comparison operators, including PL/SQL variables and constants.

Take advantage of the full range of Oracle8 Server syntax for the SELECT statement.

Remember that host variables must be prefixed with a colon.



SELECT Statements in PL/SQL

INTO clause is required.

Example

```
DECLARE
  v_deptno  NUMBER(2);
  v_loc     VARCHAR2(15);
BEGIN
  SELECT     deptno, loc
    INTO     v_deptno, v_loc
  FROM       dept
 WHERE      dname = 'SALES';
  ...
END;
```

11-177

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

INTO Clause

The INTO clause is mandatory and occurs between the SELECT and FROM clauses. It is used to specify the names of variables to hold the values that SQL returns from the SELECT clause. You must give one variable for each item selected, and their order must correspond to the items selected.

You use the INTO clause to populate either PL/SQL variables or host variables.

Queries Must Return One and Only One Row

SELECT statements within a PL/SQL block fall into the ANSI classification of Embedded SQL, for which the following rule applies:

- Queries must return one and only one row. More than one row or no row generates an error.

PL/SQL deals with these errors by raising standard exceptions, which you can trap in the exception section of the block with the NO_DATA_FOUND and TOO_MANY_ROWS exceptions (exception handling is covered in Lesson 23). You should code SELECT statements to return a single row.



Class Management Note

The INTO clause is required in a SELECT statement in PL/SQL. This is in contrast to SQL, where the INTO clause is forbidden.

Retrieving Data in PL/SQL

Retrieve the order date and the ship date for the specified order.

Example

```
DECLARE
    v_orderdate    ord.orderdate%TYPE;
    v_shipdate     ord.shipdate%TYPE;
BEGIN
    SELECT    orderdate, shipdate
      INTO    v_orderdate, v_shipdate
    FROM      ord
   WHERE     id = 157;
    ...
END;
```

11-178

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Guidelines

Follow these guidelines to retrieve data in PL/SQL:

- Terminate each SQL statement with a semicolon (;).
- The INTO clause is required for the SELECT statement when it is embedded in PL/SQL.
- The WHERE clause is optional and can be used to specify input variables, constants, literals, or PL/SQL expressions.
- Specify the same number of output variables in the INTO clause as database columns in the SELECT clause. Be sure that they correspond positionally and that their datatypes are compatible.

Class Management Note

Output variables are PL/SQL variables through which values pass from the database out to the PL/SQL block. The datatype of an output variable must be compatible with the datatype of the source column, although the source column may be an expression; in particular, Boolean variables are not permitted.

Input variables are PL/SQL variables through which values pass from the PL/SQL block into the database.

Retrieving Data in PL/SQL

Return the sum of the salaries for all employees in the specified department.

Example

```
DECLARE
    v_sum_sal      emp.sal%TYPE;
    v_deptno       NUMBER NOT NULL := 10;
BEGIN
    SELECT          SUM(sal)  -- group function
        INTO        v_sum_sal
    FROM            emp
    WHERE           deptno = v_deptno;
END;
```

11-179

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Guidelines (continued)

- To ensure that the datatypes of the identifiers match the datatypes of the columns use the %TYPE attribute. The datatype and number of variables in the INTO clause match those in the SELECT list.
- Use group functions, such as SUM, in a SQL statement, because group functions apply to groups of rows in a table.

Note: Group functions can not be used in PL/SQL syntax, they are used in SQL statements within a PL/SQL block.

Class Management Note

On the board, write:

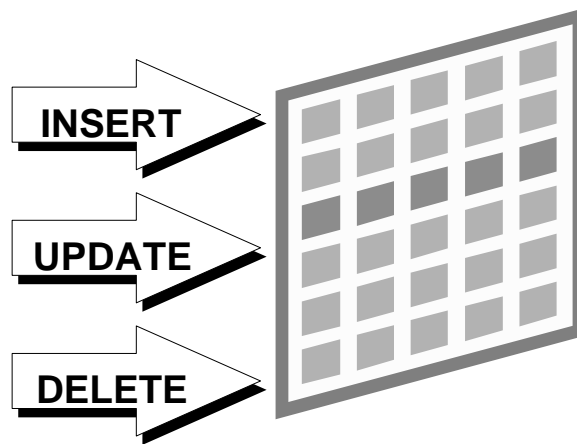
```
v_sum_salaries := SUM(s_emp.salary),
```

and draw a line through it to emphasize that group functions must be used in a SQL statement.

Manipulating Data Using PL/SQL

- **Make changes to database tables by using DML commands:**

- **INSERT**
- **UPDATE**
- **DELETE**



You manipulate data in the database by using the DML (data manipulation) commands. You can issue the DML commands INSERT, UPDATE, and DELETE without restriction in PL/SQL. By including COMMIT or ROLLBACK statements in the PL/SQL code, row locks (and table locks) are released.

- INSERT statement adds new rows of data to the table.
- UPDATE statement modifies existing rows in the table.
- DELETE statement removes unwanted rows from the table.

Inserting Data

Add new employee information to the emp table.

Example

```
DECLARE
    v_empno      emp.empno%TYPE;
BEGIN
    SELECT      empno_sequence.NEXTVAL
    INTO        v_empno
    FROM        dual;
    INSERT INTO emp(empno, ename, job, deptno)
    VALUES(v_empno, 'HARDING', 'CLERK', 10);
END;
```

11-181

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Inserting Data

- Use SQL functions, such as USER and SYSDATE.
- Generate primary key values by using database sequences.
- Derive values in the PL/SQL block.
- Add column default values.

Note: There is no possibility for ambiguity with identifiers and column names in the INSERT statement. Any identifier in the INSERT clause must be a database column name.

Class Management Note

DEMO: 118insert.sql;

PURPOSE: This example demonstrates using the INSERT command in an anonymous block.

Verify the change to the table. Enter SELECT * FROM emp where empno is the employee number entered.

Updating Data

Increase the salary of all employees in the emp table who are Analysts.

Example

```
DECLARE
  v_sal_increase    emp.sal%TYPE := 2000;
BEGIN
  UPDATE    emp
  SET       sal = sal + v_sal_increase
  WHERE     job = 'ANALYST';
END;
```

Updating and Deleting Data

There may be ambiguity in the SET clause of the UPDATE statement because although the identifier on the left of the assignment operator is always a database column, the identifier on the right can be either a database column or a PL/SQL variable.

Remember that the WHERE clause is used to determine which rows are affected. If no rows are modified, no error occurs, unlike the SELECT statement in PL/SQL.

Note: PL/SQL variable assignments always use := and SQL column assignments always use =. Recall that if column names and identifier names are identical in the WHERE clause, the Oracle8 Server looks to the database first for the name.

Class Management Note

DEMO: 118update.sql;

PURPOSE: This example demonstrates using the UPDATE command in an anonymous block.

Verify the change to the table. Enter SELECT * FROM emp.

Deleting Data

Delete rows that have belong to department 10 from the emp table.

Example

```
DECLARE
  v_emp    deptno.emp%TYPE := 10;
BEGIN
  DELETE FROM emp
    WHERE deptno = v_deptno;
END;
```

11-183

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Delete a specified order.

```
DECLARE
  v_ordid  ord.ordid%TYPE := 605;
BEGIN
  DELETE FROM item
    WHERE   ordid = v_ordid;
END;
```

Class Management Note

DEMO: 118delete.sql;

PURPOSE: This example demonstrates using the DELETE command in an anonymous block.

Verify the change to the table. Enter SELECT * FROM emp where deptno is the department number, and 'CLERK' is the job.

Naming Conventions

- Use a naming convention to avoid ambiguity in the WHERE clause.
- Database columns and identifiers should have distinct names.
- Syntax errors can arise because PL/SQL checks the database first for a column in the table.

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

Example

Retrieve the date ordered and the date shipped from the ord table where the date shipped is today. This example raises an unhandled runtime exception.

```
DECLARE
  orderdate ord.orderdate%TYPE;
  shipdate  ord.shipdate%TYPE;
  v_date DATE := SYSDATE;
BEGIN
  SELECT orderdate, shipdate
  INTO   orderdate, shipdate
  FROM   ord
  WHERE  shipdate = v_date; -- unhandled exception: TOO_MANY_ROWS
END;
SQL> /
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 6
```


Naming Conventions

```
DECLARE
  orderdate ord.orderdate%TYPE;
  shipdate  ord.shipdate%TYPE;
  v_date DATE := SYSDATE;
BEGIN
  SELECT orderdate, shipdate
  INTO   orderdate, shipdate
  FROM   ord
  WHERE  shipdate = v_date;
END;
SQL> /
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 6
```

11-185

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

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

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

The example shown in the slide is defined as follows:

- Retrieve the date ordered and the date shipped from the ord table where the date shipped is today. This example raises an unhandled runtime exception.

PL/SQL checks whether an identifier is a column in the database; if not, it is assumed to be a PL/SQL identifier.

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

For more information NO_DATA_FOUND and other exceptions, see Lesson 23.



COMMIT and ROLLBACK Commands

- **Initiate a transaction with the first DML command to follow a COMMIT or ROLLBACK.**
- **Use COMMIT and ROLLBACK SQL statements to terminate a transaction explicitly.**

Controlling Transactions

You control the logic of transactions with COMMIT and ROLLBACK SQL statements, rendering some groups of database changes permanent while discarding others. As with Oracle8 Server, DML transactions start at the first command to follow a COMMIT or ROLLBACK and end on the next successful COMMIT or ROLLBACK. These actions may occur within a PL/SQL block or as a result of events in the host environment (for example, ending a SQL*Plus session automatically commits the pending transaction).

COMMIT Command

COMMIT ends the current transaction by making all pending changes to the database permanent.

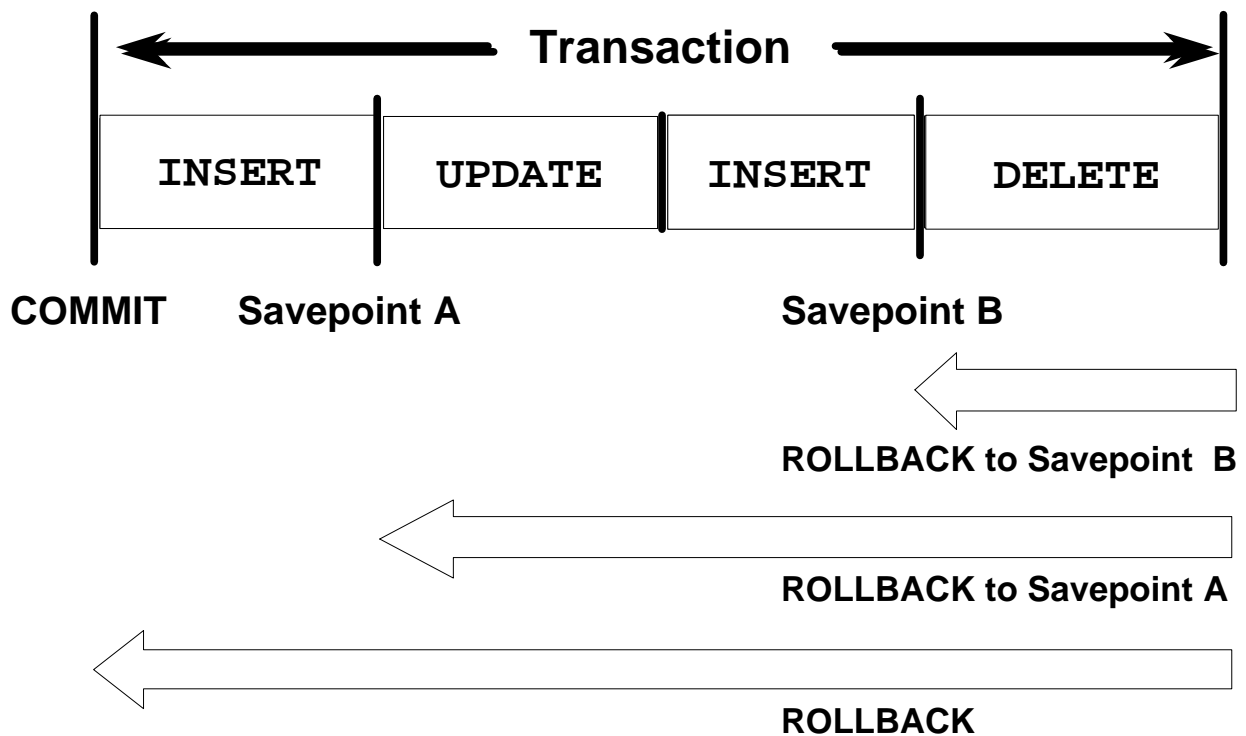
Syntax

where: WORK is for compliance with ANSI standards.

Note: The transaction control commands are all valid within PL/SQL, although the host environment may place some restriction on their use.

You can also include explicit locking commands (such as LOCK TABLE and SELECT ... FOR UPDATE) in a block (see Lesson 22 for more information on the FOR_UPDATE command). They stay in effect until the end of the transaction. Also, one PL/SQL block does not necessarily imply one transaction.

ROLLBACK Command



11-187

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

ROLLBACK ends the current transaction by discarding all pending changes.

Syntax

```
ROLLBACK [WORK] ;
```

where: WORK is for compliance with ANSI standards

Controlling Transactions

Determine the transaction processing for the following PL/SQL block.

```
BEGIN
  INSERT INTO temp(num_col1, num_col2, char_col)
    VALUES (1, 1, 'ROW 1');
  SAVEPOINT a;
  INSERT INTO temp(num_col1, num_col2, char_col)
    VALUES (2, 1, 'ROW 2');
  SAVEPOINT b;
  INSERT INTO temp(num_col1, num_col2, char_col)
    VALUES (3, 3, 'ROW 3');
  SAVEPOINT c;
  ROLLBACK TO SAVEPOINT b;
  COMMIT;
END;
```

11-188

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

SAVEPOINT Command

Alter the transactional logic with Oracle8 Server savepoints based on runtime conditions. SAVEPOINT marks an intermediate point in the transaction processing.

Syntax

where: `savepoint_name` is a PL/SQL identifier.

ROLLBACK TO SAVEPOINT discards pending changes made after the savepoint was marked.

Syntax

Note: SAVEPOINT is not ANSI standard; it is an Oracle8 Server extension of the standard SQL language. You can have more than one savepoint per transaction, each identified by a different marker name. If you create a second savepoint with the same marker name, the previous savepoint is erased. Savepoints are especially useful with conditional logic.

Class Management Note

Question: How many rows are inserted? Answer: 2

Question: Which savepoints are erased? Answer: c

Question: Which savepoints are kept? Answer: a and b

SQL Cursor

- A cursor is a private SQL work area.
- There are two types of cursors:
 - Implicit cursors
 - Explicit cursors
- The Oracle8 Server uses implicit cursors to parse and execute your SQL statements.
- Explicit cursors are explicitly declared by the programmer.

Whenever you issue a SQL statement, the Oracle8 Server opens an area of memory in which the command is parsed and executed. This area is called a *cursor*.

When the executable part of a block issues a SQL statement, PL/SQL creates an implicit cursor, which has the SQL identifier. PL/SQL manages this cursor automatically. An explicit cursor is explicitly declared and named by the programmer. There are four attributes available in PL/SQL that can be applied to cursors.

Note: For more information about explicit cursors, see Lesson 21

For more information, see

Guide and Reference, Release 8, “Interaction with Oracle.”

PL/SQL User's



SQL Cursor Attributes

Using SQL cursor attributes, you can test the outcome of your SQL statements.

SQL%ROWCOUNT	Number of rows affected by the most recent SQL statement (an integer value).
SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement affects one or more rows.
SQL%NOTFOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement does not affect any rows.
SQL%ISOPEN	Always evaluates to FALSE because PL/SQL closes implicit cursors immediately after they are executed.

11-190

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

SQL cursor attributes allow you to evaluate what happened when the implicit cursor was last used. You use these attributes in PL/SQL statements such as functions. You cannot use them in SQL statements.

You can use the attributes, SQL%ROWCOUNT, SQL%FOUND, SQL%NOTFOUND, and SQL%ISOPEN in the exception section of a block to gather information about the execution of a data manipulation statement. PL/SQL does not consider a DML statement that affects no rows to have failed, unlike the SELECT statement, which returns an exception.

Class Management Note

SQL%ISOPEN is included here for completeness; it is used with explicit cursors.

SQL Cursor Attributes

Delete rows that have the specified order number from the ITEM table. Print the number of rows deleted.

Example

```
DECLARE
  v_ordid  NUMBER := 605;
BEGIN
  DELETE FROM item
  WHERE ordid = v_ordid;
  DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT)
    || ' rows deleted. ');
END;
```

Note: Remember to enable the DBMS_OUTPUT package. You must SET SERVEROUTPUT ON in SQL*Plus.

Summary

Embed SQL in the PL/SQL block:

- **SELECT** retrieves exactly one row.
- **INSERT** adds a row.
- **UPDATE** modifies one or more existing rows.
- **DELETE** removes one or more existing rows.

Summary

Embed transaction control statements in a PL/SQL block:

- **COMMIT** makes all pending transactions permanent.
- **ROLLBACK** eliminates all pending transactions.
- **SAVEPOINT** marks an intermediate point in the transaction processing.

Summary

- **There are two cursor types; implicit and explicit.**
- **Implicit cursor attributes verify the outcome of DML statements:**
 - **SQL%ROWCOUNT**
 - **SQL%FOUND**
 - **SQL%NOTFOUND**
 - **SQL%ISOPEN**
- **Explicit cursors are defined by the user.**

Practice Overview

- **Creating a PL/SQL block to select data from a table**
- **Creating a PL/SQL block to insert data into a table**
- **Creating a PL/SQL block to update data in a table**
- **Creating a PL/SQL block to delete a record from a table**

Practice Overview

In this practice, you create procedures to select, input, update, and delete information in a table, using basic SQL query and DML statements within a PL/SQL block.

Practice 18

1. Create a PL/SQL block that selects the maximum department number in the DEPT table and store it in a SQL*Plus variable. Print the results to the screen. Save your PL/SQL block to a file named *p18q1.sql*.

```
G MAX_DEPTNO
```

```
-----  
40
```

2. Create a PL/SQL block that inserts a new department into the DEPT table. Save your PL/SQL block to a file named *p18q2.sql*.

- a. Use the department number retrieved from exercise 1 and add 10 to that number as the input department number for the new department.
- b. Create a parameter for the department name.
- c. Leave the location null for now.
- d. Execute the PL/SQL block.

```
Please enter the department number: : 50  
Please enter the department name: EDUCATION
```

```
PL/SQL procedure successfully completed.
```

- e. Display the new department that you created.

```
DEPTNO DNAME      LOC  
-----  
50 EDUCATION
```

3. Create a PL/SQL block that updates the location for an existing department. Save your PL/SQL block to a file named *p18q3.sql*.

- a. Create a parameter for the department number.
- b. Create a parameter for the department location.
- c. Test the PL/SQL block.

```
Please enter the department number: 50  
Please enter the department location: HOUSTON
```

```
PL/SQL procedure successfully completed.
```

Practice 18 (continued)

- d. Display the department number, department name, and location for the updated department.

```
DEPTNO DNAME          LOC
-----
      50 EDUCATION HOUSTON
```

4. Create a PL/SQL block that deletes the department created in exercise 2. Save your PL/SQL block to a file named *p18q4.sql*.

- Create a parameter for the department number.
- Print to the screen the number of rows affected.
- Test the PL/SQL block.

```
Please enter the department number: 50
PL/SQL procedure successfully completed.
```

```
G_RESULT
-----
1 row(s) deleted.
```

- d. What happens if you enter a department number that does not exist?

```
Please enter the department number: 99
PL/SQL procedure successfully completed.
```

```
G_RESULT
-----
0 row(s) deleted.
```

- e. Confirm that the department has been deleted.

```
no rows selected
```


11

Writing Control Structures

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	30 minutes	Lecture
	45 minutes	Practice
	75 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Identify the uses and types of control structures**
- **Construct an IF statement**
- **Construct and identify different loop statements**
- **Use logic tables**
- **Control block flow using nested loops and labels**

11-200

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Aim

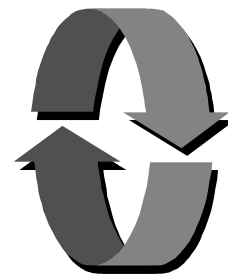
In this lesson, you will learn about conditional control within the PL/SQL block by using IF statements and loops.

Controlling PL/SQL Flow of Execution

You can change the logical flow of statements using conditional IF statements and loop control structures.

- **Conditional IF statements:**

- **IF-THEN**
- **IF-THEN-ELSE**
- **IF-THEN-ELSIF**



11-201

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

You can change the logical flow of statements within the PL/SQL block with a number of *control structures*. This lesson addresses two types of PL/SQL control structures; conditional constructs with the IF statement and and LOOP control structures (covered later in this lesson).

There are three forms of IF statements:

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSIF

Class Management Note

The GOTO statement is not addressed in this course because unconditional branching goes against the rules of structured programming. You can mention the GOTO statement, which unconditionally transfers control to a different sequence of statements. Branch to a label within the same block or to a sequence of statements, or to a label within an outer block or enclosing sequence of statements.

Example

```
BEGIN ... <<update_row>>
  BEGIN
    UPDATE emp ...
  END update_row; ...
  GOTO update_row; ...
END;
```

IF Statements

Syntax

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

Simple IF Statement:

Set the manager ID to 22 if the employee name is Osborne.

```
IF v_ename = 'OSBORNE' THEN
    v_mgr := 22;
END IF;
```

11-202

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The structure of the PL/SQL IF statement is similar to the structure of IF statements in other procedural languages. It allows PL/SQL to perform actions selectively based on conditions.

In the syntax,

condition is a Boolean variable or expression (TRUE, FALSE, or NULL). It is associated with a sequence of statements, which is executed only if the expression yields TRUE.

THEN is a clause that associates the Boolean expression that precedes it with the sequence of statements that follows it.

statements can be one or more PL/SQL or SQL statements. They may include further IF statements containing several nested IFs, ELSEs, and ELSIFs.

ELSIF is a keyword that introduces a Boolean expression. If the first condition yields FALSE or NULL then the ELSIF keyword introduces additional conditions.

ELSE is a keyword that if control reaches it, the sequence of statements that follows it is executed.

Simple IF Statements

Set the job title to Salesman, the department number to 35, and the commission to 20% of the current salary if the last name is Miller.

Example

```
. . .  
IF v_ename = 'MILLER' THEN  
    v_job := 'SALESMAN';  
    v_deptno := 35;  
    v_new_comm := sal * 0.20;  
END IF;  
. . .
```

11-203

Copyright © Oracle Corporation, 1998. All rights reserved.

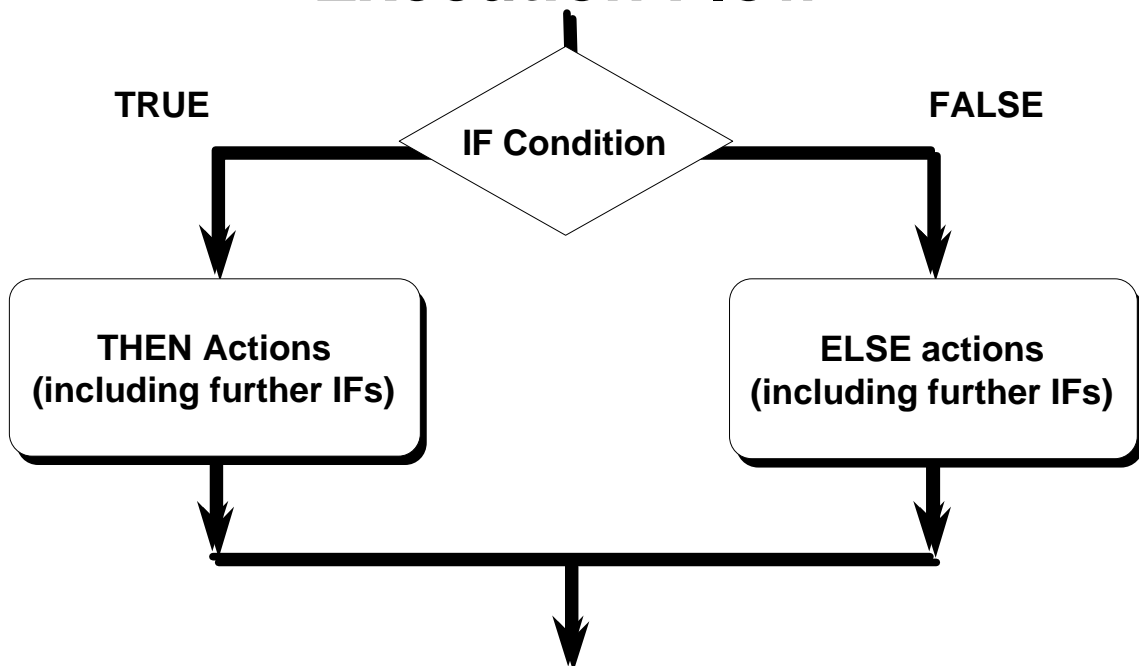
ORACLE®

In the example in the slide, PL/SQL performs these two actions (setting the `v_job`, `v_deptno`, and `v_new_comm` variables) only if the condition is TRUE. If the condition is FALSE or NULL, PL/SQL ignores them. In either case, control resumes at the next statement in the program following END IF.

Guidelines

- You can perform actions selectively based on conditions being met.
- When writing code, remember the spelling of the keywords:
 - ELSIF is one word.
 - END IF is two words.
- If the controlling Boolean condition is TRUE, the associated sequence of statements is executed; if the controlling Boolean condition is FALSE or NULL, the associated sequence of statements is passed over. Any number of ELSIF clauses is permitted.
- There can be at most one ELSE clause.
- Indent the conditionally executed statements for clarity.

IF-THEN-ELSE Statement Execution Flow



11-204

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

If the condition is FALSE or NULL, you can use the ELSE clause to carry out other actions. As with the simple IF statement, control resumes in the program from the END IF. Example:

```
IF condition1 THEN
    statement1;
ELSE
    statement2;
END IF;
```

Nested IF Statements

Either set of actions of the result of the first IF statement can include further IF statements before specific actions are performed. The THEN and ELSE clauses can include IF statements. Each nested IF statement must be terminated with a corresponding END IF.

```
IF condition1 THEN
    statement1;
ELSE
    IF condition2 THEN
        statement2;
    END IF;
END IF;
```

IF-THEN-ELSE Statements

Set a flag for orders where there are fewer than 5 days between order date and ship date.

Example

```
...  
IF v_shipdate - v_orderdate < 5 THEN  
    v_ship_flag := 'Acceptable';  
ELSE  
    v_ship_flag := 'Unacceptable';  
END IF;  
...
```

11-205

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Example

Set the job to Manager if the employee name is King. If the employee name is other than King, set the job to Clerk.

```
IF v_ename = 'KING' THEN  
    v_job := 'MANAGER';  
ELSE  
    v_job := 'CLERK';  
END IF;
```

Class Management Note

DEMO: 119ifthenelse.sql

PURPOSE: This example demonstrates using the IF-THEN-ELSE statement in an anonymous block.

Verify the change to the table. Enter SELECT comm FROM emp where empno is the employee number entered.

IF-THEN-ELSIF Statements

For a given value entered, return a calculated value.

Example

```
. . .  
IF v_start > 100 THEN  
    RETURN (2 * v_start);  
ELSIF v_start >= 50 THEN  
    RETURN (.5 * v_start);  
ELSE  
    RETURN (.1 * v_start);  
END IF;  
. . .
```

11-206

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

When possible, use the ELSIF clause instead of nesting IF statements. The code is easier to read and understand, and the logic is clearly identified. If the action in the ELSE clause consists purely of another IF statement, it is more convenient to use the ELSIF clause. This makes the code clearer by removing the need for nested END IFs at the end of each further set of conditions and actions.

Example

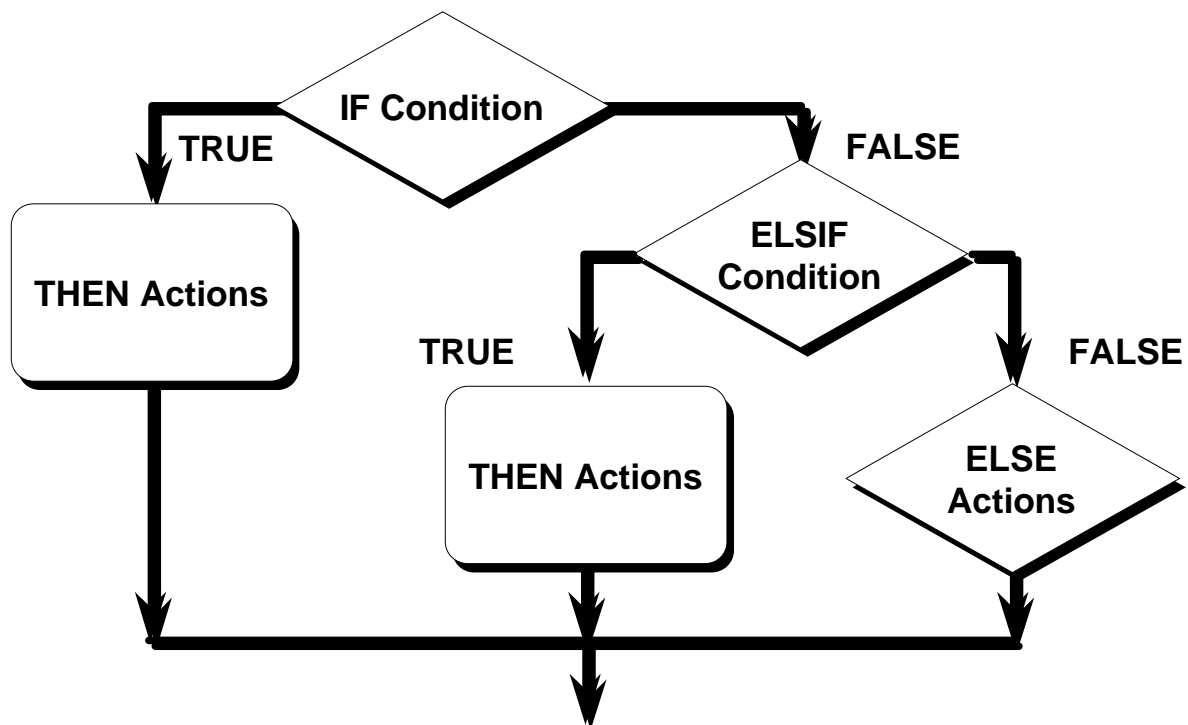
```
IF condition1 THEN  
    statement1;  
ELSIF condition2 THEN  
    statement2;  
ELSIF condition3 THEN  
    statement3;  
END IF;
```

The example IF-THEN-ELSIF statement above is further defined as follows:

For a given value entered, return a calculated value. If the entered value is over 100, then the calculated value is two times the entered value. If the entered value is between 50 and 100, then the calculated value is 50% of the starting value. If the entered value is less than 50, then the calculated value is 10% of the starting value.

Note: Any arithmetic expression containing null values evaluates to null.

IF-THEN-ELSIF Statement Execution Flow



11-207

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

In the input screen of an Oracle Forms application, enter the department number of the new employee to determine their bonus.

```
...  
IF :dept.deptno = 10 THEN  
    v_comm := 5000;  
ELSIF :dept.deptno = 20 THEN  
    v_comm := 7500;  
ELSE  
    v_comm := 2000;  
END IF;
```

In the example, the variable v_comm is used to populate a screen field with the employees bonus amount and dept.deptno represents the value entered into the screen field.

Building Logical Conditions

- You can handle null values with the **IS NULL** operator.
- Any expression containing a null value evaluates to **NULL**.
- Concatenated expressions with null values treat null values as an empty string.

11-208

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

You can build a simple Boolean condition by combining number, character, or date expressions with a comparison operator. In general, handle null values with the **IS NULL** operator.

Null in Expressions and Comparisons

- The **IS NULL** condition evaluates to **TRUE** only if the variable it is checking is **NULL**.
- Any expression containing a null value evaluates to **NULL**, with the exception of a concatenated expression, which treats the null value as an empty string.

Examples

```
v_sal > 1000
```

~~v_sal~~ evaluates to **NULL** if ~~v_sal~~ is **NULL** in both of the above examples.

~~v_sal * 1.1~~
In the next example ~~v_string~~ does not evaluate to **NULL** if ~~v_string~~ is **NULL**.

```
'PL' || v_string || 'SQL'
```


Logic Tables

Build a simple Boolean condition with a comparison operator.

AND	<i>TRUE</i>	<i>FALSE</i>	<i>NULL</i>	OR	<i>TRUE</i>	<i>FALSE</i>	<i>NULL</i>	NOT	
<i>TRUE</i>	TRUE	FALSE	NULL	<i>TRUE</i>	TRUE	TRUE	TRUE	<i>TRUE</i>	FALSE
<i>FALSE</i>	FALSE	FALSE	FALSE	<i>FALSE</i>	TRUE	FALSE	NULL	<i>FALSE</i>	TRUE
<i>NULL</i>	NULL	FALSE	NULL	<i>NULL</i>	TRUE	NULL	NULL	<i>NULL</i>	NULL

11-209

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Boolean Conditions with Logical Operators

You can build a complex Boolean condition by combining simple Boolean conditions with the logical operators AND, OR, and NOT. In the logic tables shown in the slide, FALSE takes precedence in an AND condition and TRUE takes precedence in an OR condition. AND returns TRUE only if both of its operands are TRUE. OR returns FALSE only if both of its operands are FALSE. NULL and TRUE always evaluate to NULL because it is not known if the second operand evaluates to TRUE or not.

Note: The negation of NULL (NOT NULL) results in a null value because null values are indeterminate.

Boolean Conditions

What is the value of V_FLAG in each case?

```
v_flag := v_reorder_flag AND v_available_flag;
```

V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
NULL	TRUE	NULL
NULL	FALSE	FALSE

11-210

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Building Logical Conditions

The AND logic table can help you evaluate the possibilities for the Boolean condition in the above slide.

Class Management Note

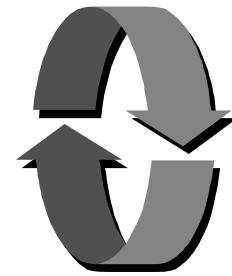
This slide uses a build feature to present the solutions in red under the column V_FLAG. Ask the students the value for V_FLAG before revealing the answer.

Answers:

1. TRUE
2. FALSE
3. NULL
4. FALSE

Iterative Control: LOOP Statements

- **Loops repeat a statement or sequence of statements multiple times.**
- **There are three loop types:**
 - **Basic loop**
 - **FOR loop**
 - **WHILE loop**



11-211

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

PL/SQL provides a number of facilities to structure loops to repeat a statement or sequence of statements multiple times.

Looping constructs are the second type of control structure:

- Basic loop to provide repetitive actions without overall conditions
- FOR loops to provide iterative control of actions based on a count
- WHILE loops to provide iterative control of actions based on a condition
- EXIT statement to terminate loops

For more information, see

PL/SQL User's Guide and Reference, Release 8, "Control Structures."



Note: Another type of FOR LOOP, cursor FOR LOOP is discussed in Lesson 22.

Basic Loop

Syntax

```
LOOP                                -- delimiter
    statement1;                    -- statements
    . . .
    EXIT [WHEN condition];        -- EXIT statement
END LOOP;                           -- delimiter
```

```
where:    condition                is a Boolean variable or
                                           expression (TRUE, FALSE,
                                           or NULL);
```

The simplest form of LOOP statement is the basic (or infinite) loop, which encloses a sequence of statements between the keywords LOOP and END LOOP. Each time the flow of execution reaches the END LOOP statement, control is returned to the corresponding LOOP statement above it. A basic loop allows execution of its statement at least once, even if the condition is already met upon entering the loop.

- Iterate through your statements with a basic loop.
- Without the EXIT statement, the loop would be infinite.

The EXIT Statement

You can terminate a loop using the EXIT statement. Control passes to the next statement after the END LOOP statement. You can issue EXIT either as an action within an IF statement, or as a standalone statement within the loop. The EXIT statement must be placed inside a loop. In the latter case, you can attach a WHEN clause to allow conditional termination of the loop. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition yields TRUE, the loop completes and control passes to the next statement after the loop. A basic loop can contain multiple EXIT statements.

For more information on the RETURN statement, see
Guide and Reference, Version 8, “Subprograms.”

PL/SQL User's



Basic Loop

Example

```
. . .  
  v_ordid      item.ordid%TYPE := 101;  
  v_counter    NUMBER(2) := 1;  
BEGIN  
. . .  
  LOOP  
    INSERT INTO item(ordid, itemid)  
      VALUES(v_ordid, v_counter);  
    v_counter := v_counter + 1;  
    EXIT WHEN v_counter > 10;  
  END LOOP;  
. . .
```

11-213

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

The basic loop example shown in the slide is defined as follows:

- Insert the first 10 new line items for order number 101.

Note: A basic loop allows execution of its statements at least once, even if the condition has been met upon entering the loop.

Class Management Note

Remind students that they can use the EXIT statement in all of these loops to set additional conditions to exit the loop.

FOR Loop

Syntax

```
FOR index IN [REVERSE]
    lower_bound..upper_bound LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

- Use a FOR loop to shortcut the test for the number of iterations.
- Do not declare the index; it is declared implicitly.

11-214

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

FOR loops have the same general structure as the Basic Loop. In addition, they have a control statement at the front of the LOOP keyword to determine the number of iterations that PL/SQL performs.

In the syntax,

index is an implicitly declared integer whose value automatically increases or decreases (decreases if the REVERSE keyword is used) by 1 on each iteration of the loop until the upper or lower bound is reached.

REVERSE causes the index to decrement with each iteration from the upper bound to the lower bound. Note that the low value (lower bound) is still referenced first.

lower_bound specifies the lower bound for the range of index values.

upper_bound specifies the upper bound for the range of index values.

Do not declare the index; it is declared implicitly as an integer.

Note: The sequence of statements is executed each time the index is incremented, as determined by the two bounds. The lower bound and upper bound of the loop range can be literals, variables, or expressions, but must evaluate to integers. If the lower bound of the loop range evaluates to a larger integer than the upper bound, the sequence of statements will not be executed.

For example, *statement1* is executed only once.

```
FOR i IN 3..3 LOOP statement1; END LOOP;
```

FOR Loop

Guidelines

- **Reference the index within the loop only; it is undefined outside the loop.**
- **Use an expression to reference the existing value of an index.**
- **Do *not* reference the index as the target of an assignment.**

11-215

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Note: The lower and upper bounds of a LOOP statement do not need to be numeric literals. They can be expressions that convert to numeric values.

Example

```
DECLARE
  v_lower    NUMBER := 1;
  v_upper    NUMBER := 100;
BEGIN
  FOR i IN v_lower..v_upper LOOP
    ...
  END LOOP;
END;
```

Class Management Note

DEMO: 119forloop.sql;

PURPOSE: This example demonstrates using the FOR LOOP command in an anonymous block.

Verify the change to the table. Enter SELECT * FROM emp where deptno is the department number.

FOR Loop

Insert the first 10 new line items for order number 101.

Example

```
. . .  
  v_ordid      item.ordid%TYPE := 101;  
BEGIN  
. . .  
  FOR i IN 1..10 LOOP  
    INSERT INTO item(ordid, itemid)  
      VALUES(v_ordid, i);  
  END LOOP;  
. . .
```

Class Management Note


Emphasize that the value of the index automatically increases by 1 on each iteration of the loop till the upper bound is reached.

Note that because “i” is such a traditional name for the index of a numeric FOR loop, the convention for naming variables with the “v_” prefix is relaxed.

WHILE Loop

Syntax

```
WHILE condition LOOP  
    statement1;  
    statement2;  
    . . .  
END LOOP;
```



Condition is evaluated at the beginning of each iteration.

Use the WHILE loop to repeat statements while a condition is TRUE.

11-217

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

You can use the WHILE loop to repeat a sequence of statements until the controlling condition is no longer TRUE. The condition is evaluated at the start of each iteration. The loop terminates when the condition is FALSE. If the condition is FALSE at the start of the loop, then no further iterations are performed.

In the syntax,

condition is a Boolean variable or expression (TRUE, FALSE, or NULL).

statement can be one or more PL/SQL or SQL statements.

If the variables involved in the conditions do not change during the body of the loop, then the condition remains TRUE and the loop does not terminate.

WHILE Loop

Example

```
ACCEPT p_price PROMPT 'Enter the price of the item: '  
ACCEPT p_itemtot PROMPT 'Enter the maximum total for  
                        purchase of item: '  
  
DECLARE  
...  
v_qty                NUMBER(8) := 1;  
v_running_total      NUMBER(7,2) := 0;  
BEGIN  
    ...  
    WHILE v_running_total < &p_itemtot LOOP  
        ...  
        v_qty := v_qty + 1;  
        v_running_total := v_qty * p_price;  
    END LOOP;  
    ...
```

11-218

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

In the example in the slide, the quantity increases with each iteration of the loop until the quantity is no longer less than the maximum price allowed for spending on the item.

Class Management Note

Point out the differences between the WHILE and numeric FOR loops. The FOR loop has an implicit counter and automatically increments each time through the loop. The WHILE loop can have an explicit counter and if so, must contain a statement to increment the counter. The WHILE loop must have an explicit condition and there must be one or more statements in the loop that perform operations that could alter the condition.

Point out in the example that it is the same request as with the basic loop, but the method is different. Also point out that there is no explicit EXIT statement. The condition determines when the loop is terminated.

Nested Loops and Labels

- **Nest loops to multiple levels.**
- **Use labels to distinguish between blocks and loops.**
- **Exit the outer loop with the EXIT statement referencing the label.**

You can nest loops to multiple levels. You can nest FOR loops within WHILE loops and WHILE loops within FOR loops. Normally the termination of a nested loop does not terminate the enclosing loop unless an exception was raised. However, you can label loops and exit the outer loop with the EXIT statement.

Label names follow the same rules as other identifiers. A label is placed before a statement, either on the same line or a separate line. Label loops by placing the label before the word LOOP within label delimiters (`<<label>>`).

If the loop is labeled, the label name can optionally be included after the END LOOP statement.

Nested Loops and Labels

```
...  
BEGIN  
  <<Outer_loop>>  
  LOOP  
    v_counter :=v_counter+1;  
    EXIT WHEN v_counter>10;  
    <<Inner_loop>>  
    LOOP  
      ...  
      EXIT Outer_loop WHEN total_done = 'YES';  
      -- Leave both loops  
      EXIT WHEN inner_done = 'YES';  
      -- Leave inner loop only  
      ...  
    END LOOP Inner_loop;  
    ...  
  END LOOP Outer_loop;  
END;
```

Summary

Change the logical flow of statements by using control structures.

- **Conditional (IF statement)**
 - **IF THEN**
 - **IF-THEN-ELSE**
 - **IF-THEN-ELSIF**

Summary

- **Loops**
 - **Basic loop**
 - **FOR loop**
 - **WHILE loop**
 - **EXIT statement**

Practice Overview

- **Performing conditional actions using the IF statement**
- **Performing iterative steps using the loop structure**

11-223

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Practice Overview

In this practice, you create procedures that incorporate loops and conditional control structures.

Practice 19

1. Run the script LABS\lab19_1.sql to create the MESSAGES table. Write a PL/SQL block to insert numbers into the MESSAGES table.
 - a. Insert the numbers 1 to 10 excluding 6 and 8.
 - b. Commit before the end of the block.
 - c. Select from the MESSAGES table to verify that your PL/SQL block worked.

RESULTS

```
-----  
1  
2  
3  
4  
5  
7  
9  
10
```

2. Create a PL/SQL block that computes the commission amount for a given employee based on the employee's salary.
 - a. Run the script LABS\lab19_2.sql to insert a new employee into the EMP table.
Note: The employee will have a NULL salary.
 - b. Accept the employee number as user input with a SQL*Plus substitution parameter.
 - c. If the employee's salary is less than \$1,000, set the commission amount for the employee to 10% of the salary.
 - d. If the employee's salary is between \$1,000 and \$1,500, set the commission amount for the employee to 15% of the salary.
 - e. If the employee's salary exceeds \$1,500, set the commission amount for the employee to 20% of the salary.
 - f. If the employee's salary is NULL, set the commission amount for the employee to 0.
 - g. Commit.
 - h. Test the PL/SQL block for each case using the following test cases, and check each updated commission.

Employee Number	Salary	Resulting Commission
7369	800	80
7934	1300	195
7499	1600	320
8000	NULL	NULL

Practice 19 (continued)

EMPNO	ENAME	SAL	COMM
8000	DOE		0
7499	ALLEN	1600	320
7934	MILLER	1300	195
7369	SMITH	800	80

If you have time, complete the following exercises.

3. Modify *p16q4.sql* to insert the text “Number is odd” or “Number is even,” depending on whether the value is odd or even, into the MESSAGES table. Query the MESSAGES table to determine if your PL/SQL block worked.

RESULTS

Number is even

4. Add a new column to the EMP table for storing asterisk (*).

```
SQL> ALTER TABLE emp
2 ADD stars VARCHAR2(50);
```

5. ~~Create a PL/SQL block that rewards an employee by appending an asterisk in the STARS~~ column for every \$100 of the employee's salary. Round the employee's salary to the nearest whole number. Save your PL/SQL block to a file called *p19q5.sql*.

- a. Accept the employee ID as user input with a SQL*Plus substitution parameter.
- b. Initialize a variable to contain a string of asterisks.
- c. Append an asterisk to the string for every \$100 of the salary amount. For example, if the employee has a salary amount of \$800, the string of asterisks should contain eight asterisks.
- d. Update the STARS column for the employee with the string of asterisks.
- e. Commit.
- f. Test the block for employees who have no salary and for an employee who has a salary.

Practice 19 (continued)

```
Please enter the employee number: 7934
PL/SQL procedure successfully completed.
Please enter the employee number: 8000
PL/SQL procedure successfully completed.
```

```
EMPNO      SAL STARS
-----
8000
7934      1300 *****
```

11

Working with Composite Datatypes

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Schedule:	Timing	Topic
	30 minutes	Lecture
	35 minutes	Practice
	65 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Create user-defined PL/SQL records**
- **Create a record with the %ROWTYPE attribute**
- **Create a PL/SQL table**
- **Create a PL/SQL table of records**
- **Describe the difference between records, tables, and tables of records**

11-228

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Aim

In this lesson, you will learn more about composite datatypes and their uses.

Composite Datatypes

- **Types:**
 - **PL/SQL RECORDS**
 - **PL/SQL TABLES**
- **Contain internal components.**
- **Are reusable.**

11-229

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

RECORDS and TABLES

Composite datatypes (also known as *collections*) are RECORD, TABLE, Nested TABLE, and VARRAY. You use the RECORD datatype to treat related but dissimilar data as a logical unit. You use the TABLE datatype to reference and manipulate collections of data as a whole object. The Nested TABLE and VARRAY datatypes are not covered in this course.

A record is a group of related data items stored in fields, each with its own name and datatype. A Table contains a column and a primary key to give you array-like access to rows. Once defined, tables and records can be reused.

For more information, see

Guide and Reference, Release 8, “Collections and Records.”

PL/SQL User's



PL/SQL Records

- **Must contain one or more components of any scalar, RECORD, or PL/SQL TABLE datatype-called fields.**
- **Are similar in structure to records in a 3GL.**
- **Are not the same as rows in a database table.**
- **Treat a collection of fields as a logical unit.**
- **Are convenient for fetching a row of data from a table for processing.**

A *record* is a group of related data items stored in *fields*, each with its own name and datatype. For example, suppose you have different kinds of data about an employee such as name, salary, hire date, and so on. This data is dissimilar in type but logically related. A record that contains such fields as the name, salary, and hire date of an employee lets you treat the data as a logical unit. When you declare a record type for these fields, they can be manipulated as a unit.

- Each record defined can have as many fields as necessary.
- Records can be assigned initial values and can be defined as NOT NULL.
- Fields without initial values are initialized to NULL.
- The DEFAULT keyword can also be used when defining fields.
- You can define RECORD types and declare user-defined records in the declarative part of any block, subprogram, or package.
- You can declare and reference nested records. A record can be the component of another record.

Creating a PL/SQL RECORD

Syntax

```
TYPE type_name IS RECORD  
    (field_declaration [, field_declaration]...);
```

Where *field_declaration* stands for

```
field_name {field_type / variable%TYPE  
            / table.column%TYPE / table%ROWTYPE}  
[[NOT NULL] {:= | DEFAULT} expr]
```

11-231

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Defining and Declaring a PL/SQL Record

To create a record, you define a RECORD type and then declare records of that type.

In the syntax,

type_name is the name of the RECORD type. This identifier is used to declare records.

field_name is the name of a field within the record.

field_type is the datatype of the field. It represents any PL/SQL datatype except REF CURSOR. You can use the %TYPE and %ROWTYPE attributes.

expr is the *field_type* or an initial value..

The NOT NULL constraint prevents the assigning of nulls to those fields. Be sure to initialize NOT NULL fields.



Creating a PL/SQL Record

Declare variables to store the name, job, and salary of a new employee.

Example

```
...  
    TYPE emp_record_type IS RECORD  
        (ename      VARCHAR2(10),  
         job        VARCHAR2(9),  
         sal        NUMBER(7,2));  
    emp_record      emp_record_type;  
...
```

11-232

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

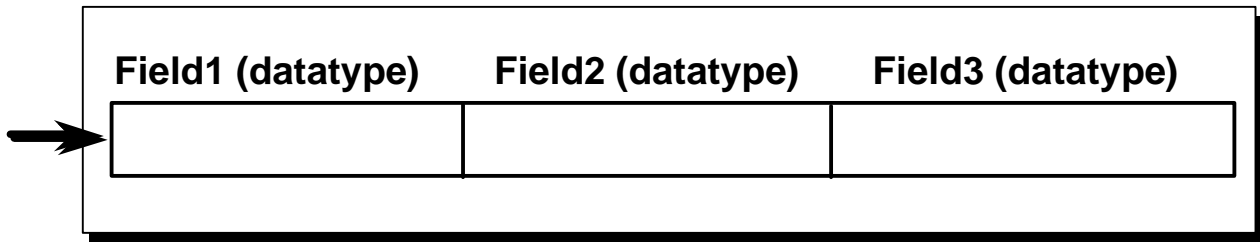
Field declarations are like variable declarations. Each field has a unique name and a specific datatype. There are no predefined datatypes for PL/SQL records, as there are for scalar variables. Therefore, you must create the datatype first and then declare an identifier using that datatype.

The following example shows that you can use the %TYPE attribute to specify a field datatype.

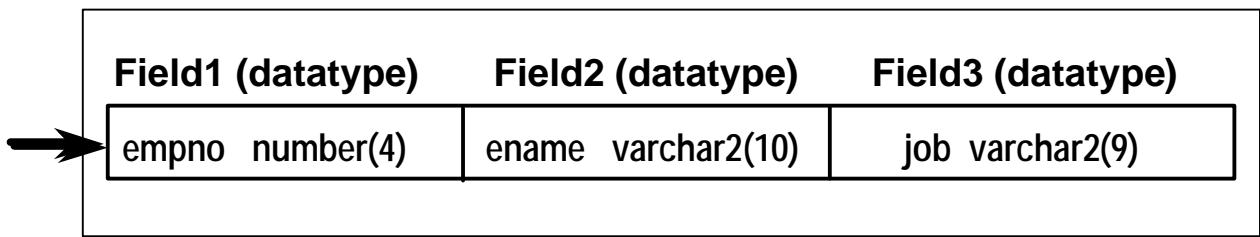
```
DECLARE  
    TYPE emp_record_type IS RECORD  
        (empno      NUMBER(4) NOT NULL := 100,  
         ename       emp.ename%TYPE,  
         job         emp.job%TYPE);  
    emp_record      emp_record_type;
```

Note: You can add the NOT NULL constraint to any field declaration and so prevent the assigning of nulls to that field. Remember, fields declared as NOT NULL must be initialized.

PL/SQL Record Structure



Example



11-233

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Referencing and Initializing Records

Fields in a record are accessed by name. To reference or initialize an individual field, you use dot notation and the following syntax:

```
record_name.field_name
```

For example, you reference field *job* in record *emp_record* as follows:

```
emp_record.job
```

You can then assign a value to the record field as follows:

```
emp_record.job := SYSDATE;
```

Assigning Values to Records

You can assign a list of common values to a record by using the SELECT or FETCH statement. Make sure that the column names appear in the same order as the fields in your record. You can also assign one record to another if they have the same datatype. A user-defined record and a %ROWTYPE record *never* have the same datatype.

The %ROWTYPE Attribute

- **Declare a variable according to a collection of columns in a database table or view.**
- **Prefix %ROWTYPE with the database table.**
- **Fields in the record take their names and datatypes from the columns of the table or view.**

11-234

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Declaring Records with the %ROWTYPE Attribute

To declare a record based upon a collection of columns in a database table or view, you use the %ROWTYPE attribute. The fields in the record take their names and datatypes from the columns of the table or view. The record can also store an entire row of data fetched from a cursor or cursor variable.

In the following example a record is declared using %ROWTYPE as a datatype specifier.

```
DECLARE  
The record, emp_record, will consist of the following fields, which reflect all of the fields that belong to the EMP  
table. emp_record emp%ROWTYPE;
```

```
(ename      VARCHAR2(10),  
 job       VARCHAR2(9),  
 sal       NUMBER,  
 comm      NUMBER)
```

Advantages of Using %ROWTYPE

- The number and datatypes of the underlying database columns may not be known.
- The number and datatypes of the underlying database column may change at runtime.
- Useful when retrieving a row with the **SELECT** statement.

11-235

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Declaring Records with the %ROWTYPE Attribute (continued)

```
DECLARE
    identifier      reference%ROWTYPE;
```

where: *identifier* is the name chosen for the record as a whole.

reference is the name of the table, view, cursor, or cursor variable on which
 the record is to be based. You must make sure that this reference is valid
when you declare the record (that is, the table or view must exist).

To reference an individual field, you use dot notation and the following syntax:

For example, you reference field *grade* in record *salary_profile* as follows:

```
record_name.field_name
```

You can then assign a value to the record field as follows:

```
emp_record.comm
```

```
emp_record.sal := 75000;
```

The %ROWTYPE Attribute

Examples

Declare a variable to store the same information about a department as it is stored in the DEPT table.

```
dept_record    dept%ROWTYPE;
```

Declare a variable to store the same information about a employee as it is stored in the EMP table.

```
emp_record    emp%ROWTYPE;
```

11-236

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Examples

The first declaration in the slide creates a record with the same field names and field datatypes as a row in the DEPT table. The fields are DEPTNO, DNAME, and LOCATION.

The second declaration above creates a record with the same field names and field datatypes as a row in the EMP table. The fields are EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, and DEPTNO.

In the following example, you select column values into a record named *item_record*.

```
DECLARE
    item_record    item%ROWTYPE;
    ...
BEGIN
    SELECT * INTO item_record
    FROM    item
```

Class Management Note

Draw on the board the structure of ITEM_RECORD.

PL/SQL Tables

- **Are composed of two components:**
 - **Primary key of datatype `BINARY_INTEGER`**
 - **Column of scalar or record datatype**
- **Increase dynamically because they are unconstrained**

11-237

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Objects of type `TABLE` are called PL/SQL tables. They are modeled as (but not the same as) database tables. PL/SQL tables use a primary key to give you array like access to rows.

A PL/SQL table

- Is similar to an array.
- Must contain two components:
 - A primary key of datatype `BINARY_INTEGER` that indexes the PL/SQL `TABLE`.
 - A column of a scalar or record datatype, which stores the PL/SQL `TABLE` elements.
- Can increase dynamically because it is unconstrained.

Note: The PL/SQL table of records is covered on page 20-12 of this lesson.

Creating a PL/SQL Table

Syntax

```
TYPE type_name IS TABLE OF  
    {column_type | variable%TYPE  
    | table.column%TYPE} [NOT NULL]  
    INDEX BY BINARY_INTEGER;  
identifier type_name;
```

Declare a PL/SQL variable to store a name.

Example

```
...  
TYPE ename_table_type IS TABLE OF emp.ename%TYPE  
    INDEX BY BINARY_INTEGER;  
ename_table ename_table_type;  
...
```

11-238

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Declaring PL/SQL Tables

There are two steps involved in creating a PL/SQL table.

1. Declare a TABLE datatype.
2. Declare a variable of that datatype.

In the syntax,

type_name is the name of the TABLE type. It is a type specifier used in subsequent declarations of PL/SQL tables.

column_type is any scalar (not composite) datatype such as VARCHAR2, DATE, or NUMBER. You can use the %TYPE attribute to provide the column datatype.

identifier is the name of the identifier which represents an entire PL/SQL table.

The NOT NULL constraint prevents nulls from being assigned to the PL/SQL TABLE of that type. Do not initialize the PL/SQL TABLE.

Declare a PL/SQL variable to store the date.

```
DECLARE  
    TYPE date_table_type IS TABLE OF DATE  
        INDEX BY BINARY_INTEGER;  
    date_table date_table_type;
```

PL/SQL Table Structure

Primary Key

...
1
2
3
...

BINARY_INTEGER

Column

...
Jones
Smith
Maduro
...

Scalar

Like the size of a database table, the size of a PL/SQL table is unconstrained. That is, the number of rows in a PL/SQL table can increase dynamically, so your PL/SQL table grows as new rows are added.

PL/SQL tables can have one column and a primary key, neither of which can be named. The column can belong to any scalar or record data type, but the primary key must belong to type BINARY_INTEGER. You cannot initialize a PL/SQL table in its declaration.

Creating a PL/SQL TABLE

```
DECLARE
  TYPE ename_table_type IS TABLE OF emp.ename%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY BINARY_INTEGER;
  ename_table      ename_table_type;
  hiredate_table   hiredate_table_type;
BEGIN
  ename_table(1) := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
  IF ename_table.EXISTS(1) THEN
    INSERT INTO ...
    ...
END;
```

11-240

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

There are no predefined datatypes for PL/SQL records, as there are for scalar variables. Therefore you must create the datatype first and then declare an identifier using that datatype.

Referencing a PL/SQL table

Syntax

```
pl/sql_table_name(primary_key_value)
where: primary_key_value belongs to type BINARY_INTEGER.
```

Reference the third row in a PL/SQL table `ename_table`.

The magnitude range of a `BINARY_INTEGER` is `-2147483647 .. 2147483647`, so the primary key value can be negative and does not start with 1.

Note: The `table.EXISTS(i)` statement returns TRUE if at least one row with index *i* is returned. Use the EXISTS statement to prevent an error which is raised in reference to a non-existing table element.

PL/SQL TABLE of RECORDS

- Define a TABLE variable with the %ROWTYPE attribute.
- Declare a PL/SQL variable to hold department information.

Example

```
DECLARE
  TYPE dept_table_type IS TABLE OF dept%ROWTYPE
    INDEX BY BINARY INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

11-241

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Because only one table definition is needed to hold information about all of the fields of a database table, the table of records greatly increases the functionality of PL/SQL tables.

Referencing a Table of Records

In the example given in the slide, you can refer to fields in the dept_table record because each element of this table is a record.

Syntax

```
table(index).field
```

Example

Location represents a field in the DEPT table.

```
dept_table(15).location := 'Atlanta';
```

Note: You can use the %ROWTYPE attribute to declare a record that represents a row in a database table. The difference between the %ROWTYPE attribute and the composite datatype RECORD is that RECORD allows you to specify the datatypes of fields in the record or to declare fields of your own.

Summary

Define and reference PL/SQL variables of composite datatypes:

- **PL/SQL Records**
- **PL/SQL Tables**
- **PL/SQL Table of Records**

Define a PL/SQL Record using the %ROWTYPE attribute.

Practice Overview

- **Declaring PL/SQL tables**
- **Processing data using PL/SQL tables**

11-243

Copyright © Oracle Corporation, 1998. All rights reserved.

ORACLE®

Practice Overview

In this practice, you define, create, and use PL/SQL tables.

Practice 20

1. Run the script LABS\lab20_1.sql to create a new table for storing employees and their salaries.

```
SQL> CREATE TABLE      top_dogs
  2  (name      VARCHAR2(25),
  3  salary      NUMBER(11,2));
```

2. Write a PL/SQL block to retrieve the name and salary of a given employee from the EMP table based on the employee's number, incorporate PL/SQL tables.
 - a. Declare two PL/SQL tables, ENAME_TABLE and SAL_TABLE, to temporarily store the names and salaries.
 - b. As each name and salary is retrieved within the loop, store them in the PL/SQL tables.
 - c. Outside the loop, transfer the names and salaries from the PL/SQL tables into the TOP_DOGS table.
 - d. Empty the TOP_DOGS table and test the practice.

```
Please enter the employee number: 7934
PL/SQL procedure successfully completed.
```

NAME	SALARY
-----	-----
MILLER	1300

```
Please enter the employee number: 7876
PL/SQL procedure successfully completed.
```

NAME	SALARY
-----	-----
ADAMS	1100