Oracle9i Database

Oracle9i Database manages all your data. It can also be unstructured data like:
• Spreadsheets
• Word documents
• Powerpoint presentations
• XML
• Multimedia data types like MP3, graphics, video, and more
The data does not even have to be in the database. Oracle9i Database has services through which you can store metadata about information stored in file systems. You can use the database server to manage and serve information wherever it is located.

System Development Life Cycle

System Development Life Cycle
From concept to production, you can develop a database by using the system development life cycle, which contains multiple stages of development. This top-down, systematic approach to database development transforms business information requirements into an operational database.

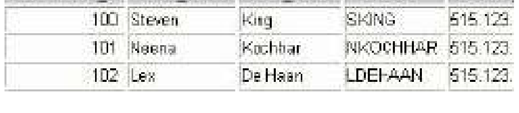
Strategy and Analysis
• Study and analyze the business requirements. Interview users and managers to identify the information requirements. Incorporate the enterprise and application mission statements as well as any future system specifications.
• Build models of the system. Transfer the business narrative into a graphical representation of business information needs and rules. Confirm and refine the mod el with the analysts and experts.

Design
Design the database based on the model developed in the strategy and analysis phase.

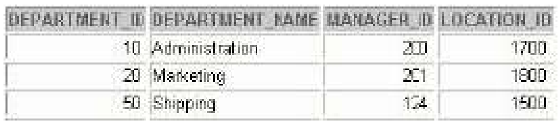
Build and Document
• Build the prototype system. Write and execute the commands to create the tables and supporting objects for the database.
• Develop user documentation, Help text, and operations manuals to support the use and operation of the system.

Relational Database Concept

• Dr. E.F. Codd proposed the relational model for **database systems in 1970.**
• It is the basis for the relational database **management system.**
• The relational model consists of the following:
- Collection of objects or relations
- Set of operators to act on the relations
– Data integrity for accuracy and consistency



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE
100	Steven	King	SKING	515.123.
101	Neena	Kochhar	NKOCHHAR	515.123.
102	Lex	De Haan	LDEHAAN	515.123.



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables.

Oracle server
Table Name: EMPLOYEES Table Name: DEPARTMENTS

Definition of a Relational Database
A relational database uses relations or two-dimensional tables to store information.
For example, you might want to store information about all the employees in your company. In a relational database, you create several tables to store different pieces of information about your employees, such as an employee table, a department table, and a salary table.

ER Modeling
In an effective system, data is divided into discrete categories or entities. An entity relationship (ER) model is an illustration of various entities in a business and the relationships between them. An ER model is derived from business specifications or narratives and built during the analysis phase of the system development life cycle. ER models separate the information required by a business from the activities performed within a business. Although businesses can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant.

Key Components
• Entity: A thing of significance about which information needs to be kno wn. Examples are departments, employees, and orders.
• Attribute: Something that describes or qualifies an entity. For example, for the employee entity, the attributes would be the employee number, name, job title, hire date, department number, and so on. Each of the attributes is either required or optional. This state is called optionality.
• Relationship: A named association between entities showing optionality and degree. Examples are employees and departments, and orders and items.

Unique Identifiers
A unique identifier (UID) is any combination of attributes or relationships, or both, that serves to distinguish occurrences of an entity. Each entity occurrence must be uniquely identifiable.
• Tag each attribute that is part of the UID with a number symbol: #
• Tag secondary UIDs with a number sign in parentheses: (#)

Relating Multiple Tables

• Each row of data in a table is uniquely **identified by a primary key (PK).**
• You can logically relate data from multiple **tables using foreign keys (FK).**

Relating Multiple Tables
Each table contains data that describes exactly one entity. For example, the EMPLOYEES table contains information about employees. Categories of data are listed across the top of each table, and individual cases are listed below. Using a table format, you can readily visualize, understand, and use information.Because data about different entities is stored in different tables, you may need to combine two or more tables to answer a particular question. For example, you may want to know the location of the department where an employee works. In this scenario, you need information from the EMPLOYEES table (which contains data about employees) and the DEPARTMENTS table (which contains information about departments). With an RDBMS you can relate the data in one table to the data in another by using the foreign keys. A foreign key is a column or a set of columns that refer to a primary key in the same table or another table.You can use the ability to relate data in one table to data in another to organize information in separate, manageable units. Employee data can be kept logically distinct from department data by storing it in a separate table.

Guidelines for Primary Keys and Foreign Keys
• You can not use duplicate values in a primary key.
• Primary keys generally cannot be changed.
• Foreign keys are based on data values and are purely logical, not physical, pointers.
• A foreign key value must match an existing primary key value or unique key value, or else be null.
• A foreign key must reference either a primary key or unique key column.

Relational Database Properties

A relational database:
• Can be accessed and modified by executing **structured query language (SQL) statements**
• Contains a collection of tables with no physical **pointers**
• Uses a set of operators

Properties of a Relational Database
In a relational database, you do not specify the access route to the tables, and you do not need to know how the data is arranged physically.
To access the database, you execute a structured query language (SQL) statement, which is the American National Standards Institute (ANSI) standard language for operating relational databases. The language contains a large set of operators for partitioning and combining relations. The database can be modified by using SQL statements.

Communicating with a RDBMS Using SQL

SQL statement is entered.

Statement is sent to Oracle Server.

SELECT department_name FROM departments;

Data is displayed.

Some SQL Statements you need to be familiar with

SELECT for Data retrieval

INSERT UPDATE DELETE for Data maintenance

CREATE ALTER DROP RENAME for Tables and other objects maintenance

TRUNCATE COMMIT ROLLBACK SAVEPOINT for Transaction Management

GRANT REVOKE for user access

Writing Basic SQL SELECT Statements

Capabilities of SQL SELECT Statements

A SELECT statement retrieves information from the database. Using a SELECT statement, you can do the
following:

• Projection: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many
columns of the table as you require.

• Selection: You can use the selection capability in SQL to choose the rows in a table that you want returned by a query. You can use various criteria to restrict the rows that you see.

• Joining: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link between them.

Basic SELECT Statement

• SELECT identifies what columns

• FROM identifies which table

Basic SELECT Statement

In its simplest form, a SELECT statement must include the following:

• A SELECT clause, which specifies the columns to be displayed

• A FROM clause, which specifies the table containing the columns listed in the SELECT clause In the syntax:

SELECT is a list of one or more columns

DISTINCT suppresses duplicates

FROM table specifies the table containing the columns

The following SQL statement, like the example on the slide, displays all columns and all rows of the DEPARTMENTS table:

SELECT department_id, department_name, manager_id, location_ id FROM departments;

Writing SQL Statements

• SQL statements are not case sensitive.
• SQL statements can be on one or more lines.
• Keywords cannot be abbreviated or split **across lines.**
• Clauses are usually placed on separate lines.
• Indents are used to enhance readability.

Executing SQL Statements

Using iSQL*Plus, click the Execute button to run the command or commands in the editing window.
In iSQL*Plus, column headings are displayed in uppercase and centered.
SELECT last_name, hire_date, salary FROM   employees;
You can override the column heading display with an alias.

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators.
You can use arithmetic operators in any clause of a SQL statement except in the FROM clause.

Using Arithmetic Operators

SELECT last_name, salary, salary + 300 FROM   employees;
• Multiplication and division take priority over **addition and subtraction.**
• Operators of the same priority are evaluated from **left to right.**
• Parentheses are used to force prioritized **evaluation and to clarify statements.**

Defining a Null Value

• A null is a value that is unavailable, unassigned, **unknown, or inapplicable.**
• A null is not the same as zero or a blank space.**Null Values**

If a row lacks the data value for a particular column, that value is said to be null, or to contain a null. A null is a value that is unavailable, unassigned, unknown, or inapplicable. A null is not the same as zero or a space. Zero is a number, and a space is a character. Columns of any data type can contain nulls. However,
some constraints, NOT NULL and PRIMARY KEY, prevent nulls from being used in the column.
In the COMMISSION_PCT column in the EMPLOYEES table, notice that only a sales manager or sales representative can earn a commission. Other employees are not entitled to earn commissions. A null represents that fact.

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.
Defining a Column Alias

A column alias:
• Renames a column heading
• Is useful with calculations
• Immediately follows the column name: there can **also be the optional AS keyword**

between the column name and alias
• Requires double quotation marks if it contains **spaces or special characters or is case sensitive**

Column Aliases
Specify the alias after the column in the SELECT list using a space as a separator. By default, alias headings appear in uppercase. If the alias contains spaces or special characters (such as # or $), or is case sensitive, enclose the alias in double quotation marks (" ").

Using Column Aliases

SELECT last_name "Name", salary*12 "Annual Salary" FROM employees;
SELECT last_name AS name, commission_pct comm FROM employees;

Concatenation Operator
A concatenation operator:
• Concatenates columns or character strings to **other columns**
• Is represented by two vertical bars (||)
• Creates a resultant column that is a character **expression**
1-18 Copyright © Oracle Corporation, 2001. All rights reserved.
Concatenation Operator
You can link columns to other columns, arithmetic expressions, or constant values to create a character expression by using the concatenation operator (||). Columns on either side of the operator are combined to make a single output column.

Using the Concatenation Operator
SELECT last_name||job_id AS "Employees" FROM employees;
Literal Character Strings
• A literal value is a character, a number, or a date **included in the SELECT list.**
• Date and character literal values must be enclosed **within single quotation marks.**
• Each character string is output once for each **row returned.**

Using Literal Character Strings
SELECT last_name ||' is a '||job_id AS "Employee Details" FROM employees;
Duplicate Rows
The default display of queries is all rows, including duplicate rows.
SELECT department_id FROM employees;
Eliminating Duplicate Rows
Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.

SELECT DISTINCT department_id FROM employees;

You can specify multiple columns after the DISTINCT qualifier. The DISTINCT qualifier affects all the selected columns, and the result is every distinct combination of the columns.

SELECT DISTINCT department_id, job_id FROM employees;

Restricting and Sorting Data

Objectives

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

• Limit the rows retrieved by a query

• Sort the rows retrieved by a query

Limiting Rows Using a Selection

Assume that you want to display all the employees in department 90. This method of restriction is the basis of the WHERE clause in SQL.

You can restrict the rows returned from the query by using the WHERE clause. A WHERE clause contains a condition that must be met, and it directly follows the FROM clause. If the condition is true, the row meeting the condition is returned. The WHERE clause can compare values in columns, literal values, arithmetic expressions, or functions. It consists of three elements:

• Column name

• Comparison condition

• Column name, constant, or list of values

Using the WHERE Clause

SELECT employee_id, last_name, job_id, department_id FROM employees WHERE department_id = 90;

Character Strings and Dates

Character strings and dates in the WHERE clause must be enclosed in single quotation marks (''). Number constants, however, should not be enclosed in single quotation marks. All character searches are case sensitive. Oracle databases store dates in an internal numeric format, representing the century, year, month, day,
hours, minutes, and seconds. The default date display is DD-MON-RR.

Comparison Conditions

Operator = > >= < <= <>

Example

... WHERE hire_date='01-JAN-95'

... WHERE salary>=6000

... WHERE last_name='Smith'

An alias cannot be used in the WHERE clause.

Note: The symbol != and ^= can also represent the not equal to condition.

Using Comparison Conditions

SELECT last_name, salary FROM employees WHERE salary <= 3000;

Using the Comparison Conditions

In the example, the SELECT statement retrieves the last name and salary from the EMPLOYEES table, where the employee salary is less than or equal to $3000. Note that there is an explicit value supplied to the WHERE clause. The explicit value of $3000 is compared to the salary value in the SALARY column of the EMPLOYEES table.

Other Comparison Conditions
BETWEEN ...AND...
IN(set)
LIKE
IS NULL
Using the BETWEEN Condition
Use the BETWEEN condition to display rows based on a range of values.
SELECT last_name, salary FROM employees WHERE salary BETWEEN 2500 AND 3500;
Using the IN Condition
Use the IN membership condition to test for values in a list.
SELECT employee_id, last_name, salary, manager_id FROM employees WHERE manager_id IN (100, 101, 201);
If characters or dates are used in the list, they must be enclosed in single quotation marks ('').
Using the LIKE Condition
• Use the LIKE condition to perform wildcard **searches of valid search string values.**
• Search conditions can contain either literal **characters or numbers:**
– % denotes zero or many characters. - _ denotes one character.
–
SELECT first_name FROM employees WHERE first_name LIKE 'S%';
• You can combine pattern-matching characters.

SELECT last_name FROM employees WHERE last_name LIKE '_o%';
• You can use the ESCAPE identifier to search for the **actual % and _ symbols.**

The ESCAPE Option
When you need to have an exact match for the actual % and _ characters, use the ESCAPE option. This option specifies what the escape character is. If you want to search for strings that contain SA_, you can search for it using the following SQL statement:
SELECT employee_id, last_name, job_id FROM employees WHERE job_id LIKE '%SA_%' ESCAPE '\';
The ESCAPE option identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (_). This causes the Oracle Server to interpret the underscore literally.

Using the NULL Conditions
SELECT last_name, manager_id FROM employees WHERE manager_id IS NULL;
The NULL conditions include the IS NULL condition and the IS NOT NULL condition.
you cannot test with = because a null cannot be equal or unequal to any value.

Logical Conditions
AND Returns TRUE if both component conditions are true
OR Returns TRUE if either component condition is true
NOT Returns TRUE if the following condition is false
Using the AND Operator
SELECT employee_id, last_name, job_id, salary FROM employees WHERE salary >=10000 AND job_id LIKE '%MAN%';
Using the OR Operator
SELECT employee_id, last_name, job_id, salary FROM employees WHERE salary >= 10000 OR job_id LIKE '%MAN%';
Using the NOT Operator

SELECT last_name, job_id FROM employees WHERE job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');

Note: The NOT operator can also be used with other SQL operators, such as BETWEEN, LIKE, and NULL.

... WHERE job_id NOT IN ('AC_ACCOUNT', 'AD_VP')
... WHERE salary NOT BETWEEN 10000 AND 15000
... WHERE last_name NOT LIKE '%A%'
... WHERE commission_pct IS NOT NULL

Rules of Precedence

1 **Arithmetic operators**
2 **Concatenation operator**
3 **Comparison conditions**
4 **IS [NOT] NULL, LIKE, [NOT] IN**
5 **[NOT] BETWEEN**
6 **NOT logical condition**
7 **AND logical condition**
8 **OR logical condition**

Override rules of precedence by using parentheses.

Rules of Precedence

SELECT last_name, job_id, salary FROM employees WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;

the SELECT statement reads as follows:
"Select the row if an employee is a president and earns more than $15,000, or if the employee is a sales representative."

Use parentheses to force priority.

SELECT last_name, job_id, salary FROM employees WHERE (job_id = 'SA_REP' OR job_id = 'AD_PRES')

the SELECT statement reads as follows:
"Select the row if an employee is a president or a sales representative, and if the employee earns more than $15,000."

ORDER BY Clause

• Sort rows with the ORDER BY clause
- ASC: ascending order (the default order)
- DESC: descending order
• The ORDER BY clause comes last in the SELECT **statement.**

SELECT last_name, job_id, department_id, hire_date FR
OM employees
ORDER BY hire_date;

If you use the ORDER BY clause, it must be the last clause of the SQL statement. You can specify an
expression, or an alias, or column position as the sort condition.

Default Ordering of Data

The default sort order is ascending:
• Numeric values are displayed with the lowest values first: for e xample, 1-999.
• Date values are displayed with the earliest value first: for exa mple, 01-JAN-92 before 01-JAN -95.
• Character values are displayed in alphabetical order: for example, A first and Z last.
• Null values are displayed last for ascending sequences and first for descending sequences.

Sorting by Multiple Columns
• The order of ORDER BY list is the order of sort.
SELECT last_name, department_id, salary FROM employees
ORDER BY department_id, salary DESC;
• You can sort by a column that is not in the **SELECT list.**

Single-Row Functions
Functions make the basic query block more powerful and are used to manipulate data values. This focuses on single-row character, number, and date functions,
as well as those functions that convert data from one type to an other: for example, character data to numeric data.
Functions are a very powerful feature of SQL and can be used to do the following:
• Perform calculations on data
• Modify individual data items
• Manipulate output for groups of rows
• Format dates and numbers for display
• Convert column data types
SQL functions sometimes take arguments and always return a value.
Two Types of SQL Functions
Single-row Multiple-row
Single-Row Functions
These functions operate on single rows only and return one result per row.
Multiple-Row Functions
Functions can manipulate groups of rows to give one result per group of rows. These functions are known as group functions.

Single-row functions are used to manipulate data items. They accept one or more arguments and return one value for each row returned by the query. An argument can be one of the following:
• User-supplied constant
• Variable value
• Column name
• Expression
• Can be used in SELECT, WHERE, and ORDER BY clauses; can be nested
This lesson covers the following single -row functions:
• Character functions: ccept character input and can return both character and number values
• Number functions: Accept numeric input and return numeric values
• Date functions: Operate on values of the DATE data type (All date functions return a value of DATE data type except the MONTHS_BETWEEN function, which returns a number.)
• Conversion functions: Convert a value from one data type to another
• General functions:
- NVL
- NVL2
- NULLIF
- COALSECE
- CASE
– DECODE
–
Character Functions
LOWER UPPER INITCAP CONCAT SUBSTR LENGTH INSTR LPAD | RPAD TRIM REPLACE

Purpose

LOWER Converts alpha character values to lowercase
UPPER Converts alpha character values to uppercase
INITCAP Converts alpha character values to uppercase for the first letter of each word, all other letters in lowercase
CONCAT Concatenates the first character value to the second character value; equivalent to concatenation operator (||)
SUBSTR(column|expression,m[,n]) Returns specified characters from character value starting at character position m, n characters long (If m is negative, the
count starts from the end of the character value. If n is omitted, all characters to the end of the string are returned.)
LENGTH Returns the number of characters in the expression
INSTR(column|expression, 'string', [,m], [n]) Returns the numeric position of a named string. Optionally, you can provide a position m to start searching, and the
occurrence n of the string. m and n default to 1, meaning start the search at the beginning of the search and report the first occurrence.
LPAD(column|expression, n, 'string')
RPAD(column|expression, n, 'string') Pads the character value left/right-justified to a total width of n character positions
TRIM(leading|trailing|both , trim_character FROM trim_source) Enables you to trim heading or trailing characters (or both) from a character string. If trim_character or *trim_source is a character literal, you must enclose it in single quotes.*
REPLACE(text, search_string, replacement_string) Searches a text expression for a character string and, if found, replaces it with a specified replacement string

Character-Manipulation Functions
These functions manipulate character strings:

Function	**Result**
CONCAT('Hello', 'World')	**HelloWorld**
SUBSTR('HelloWorld',1,5)	**Hello**
LENGTH('HelloWorld')	**10**
INSTR('HelloWorld', 'W')	**6**
LPAD(salary,10,'*')	*******24000**
RPAD(salary, 10, '*')	**24000*******
TRIM('H' FROM 'HelloWorld')	**elloWorld**

Number Functions

• ROUND: Rounds value to specified decimal **ROUND(45.926, 2) 45.93**
• TRUNC: Truncates value to specified decimal **TRUNC(45.926, 2) 45.92**
• MOD: Returns remainder of division **MOD(1600, 300) 100**

ROUND Function

The ROUND function rounds the column, expression, or value to n decimal places. If the second argument
is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is
rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two
decimal places to the left.
The ROUND function can also be used with date functions.

The DUAL Table
The DUAL table is owned by the user SYS and can be accessed by all users. It contains one column, DUMMY, and one row with the value X. The DUAL table is useful when you want to return a value once only: for instance, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data. The DUAL table is generally used for SELECT clause syntax completeness, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from actual tables.

TRUNC Function
The TRUNC function truncates the column, expression, or value to n decimal places.
The TRUNC function works with arguments similar to those of the ROUND function. If the second argument is 0 or is missing, the value is truncated to zero decimal places. If the second argument is 2, the value is truncated to two decimal places. Conversely, if the second argument is -2, the value is truncated to two decimal places to the left.
Like the ROUND function, the TRUNC function can be used with date functions.

MOD Function
The MOD function finds the remainder of value1 divided by value2. **The MOD function is often used to determine if a value is odd or even.**

<u>Working with Dates</u>

• Oracle database stores dates in an internal **numeric format: century, year, month, day, hours, minutes, seconds.**
• The default date display format is DD-MON-RR.

Oracle Date Format
Valid Oracle dates are between January 1, 4712 B.C. and A.D. December 31, 9999.
This data is stored internally as follows:

CENTURY	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
19	94	06	07	5	10	43

The SYSDATE Function
SYSDATE is a date function that returns the current database server date and time. You can use SYSDATE just as you would use any other column name. For example, you ca n display the current date by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL.

Arithmetic with Dates

Arithmetic with Dates

Because the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates.

Operation	**Result**	**Description**
date + number	Date	Adds a number of days to a date
date - number	Date	Subtracts a number of days from a date
date - date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

If a more current date is subtracted from an older date, the difference is a negative number.

Date Functions

Date functions operate on Oracle dates. All date functions return a value of DATE data type except MONTHS_BETWEEN, which returns a numeric value.

• MONTHS_BETWEEN(date1, date2): Finds the number of months between date1 and date2. The result can be positive or negative. If date1 is later than date2, the result is positive; if date1 is earlier than date2, the result is negative. The noninteger part of the result represents a portion of the month.

• ADD_MONTHS(date, n): Adds n number of calendar months to date. The value of n must be an integer and can be negative.

• NEXT_DAY(date, 'char'): Finds the date of the next specified day of the week ('char') following date. The value of char may be a number representing a day or a character string.

• LAST_DAY(date): Finds the date of the last day of the month that contains date.

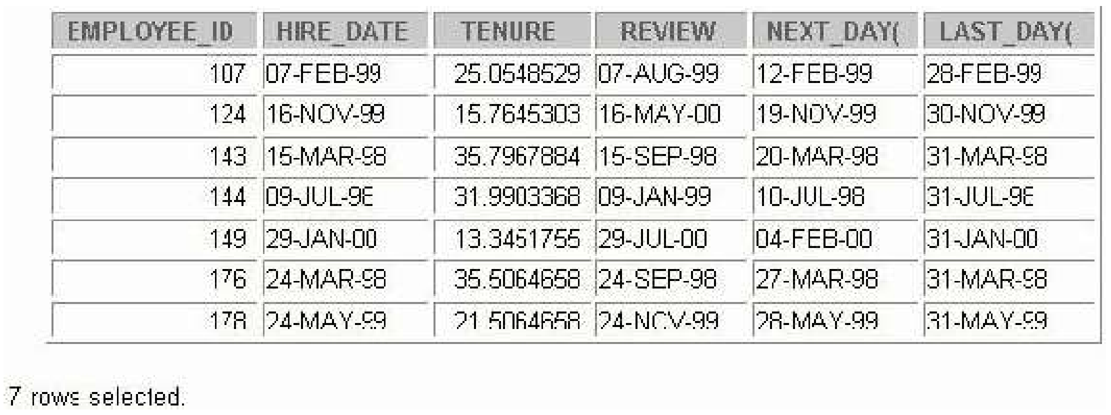
• ROUND(date[,'fmt']): Returns date rounded to the unit specified by the format model fmt. If the format model fmt is omitted, date is rounded to the nearest day.

• TRUNC(date[, 'fmt']): Returns date with the time portion of the day truncated to the unit specified by the format model fmt. If the format model fmt is omitted, date is truncated to the nearest day.

• MONTHS_BETWEEN ('01-SEP-95','11-JAN-94') **19.6774194**
• ADD_MONTHS ('11-JAN-94',6) **'11-JUL-94'**
• NEXT_DAY ('01-SEP-95','FRIDAY') **'08-SEP-95'**
• LAST_DAY('01-FEB-95') **'28-FEB-95'**

Assume SYSDATE = '25-JUL-95':

• ROUND(SYSDATE,'MONTH')	01-AUG-95
• ROUND(SYSDATE ,'YEAR')	01-JAN-96
• TRUNC(SYSDATE ,'MONTH')	01-JUL-95
• TRUNC(SYSDATE ,'YEAR')	01-JAN-95

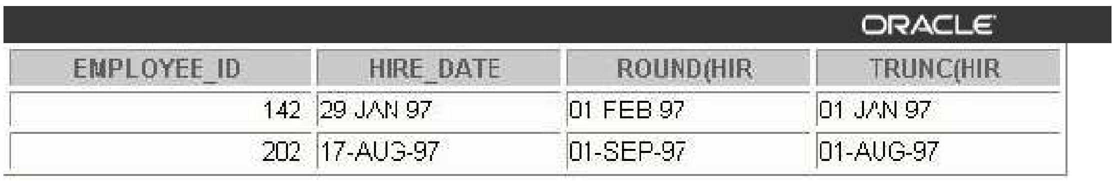


EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(	LAST_DAY(
107	07-FEB-99	25.0548529	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	15.7645303	16-MAY-00	19-NOV-99	30-NOV-99
143	15-MAR-98	35.7967884	15-SEP-98	20-MAR-98	31-MAR-98
144	09-JUL-98	31.9903368	09-JAN-99	10-JUL-98	31-JUL-98
149	29-JAN-00	13.3451755	29-JUL-00	04-FEB-00	31-JAN-00
176	24-MAR-98	35.5064658	24-SEP-98	27-MAR-98	31-MAR-98
178	24-MAY-99	21.5064658	24-NOV-99	28-MAY-99	31-MAY-99

```
7 rows selected.
```

For example, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and last day of the month when hir ed for all employees employed for fewer than 36 months.

```
SELECT employee_id, hire_date, MONTHS_BETWEEN (SYSDATE, hire_date)
TENURE,ADD_MONTHS (hire_date, 6) REVIEW, NEXT_DAY (hire_date, 'FRIDAY'),
LAST_DAY(hire_date)
FROM   employees
WHERE  MONTHS_BETWEEN (SYSDATE, hire_date) < 36;
```



EMPLOYEE_ID	HIRE_DATE	ROUND(HIR	TRUNC(HIR
142	29-JAN-97	01-FEB-97	01-JAN-97
202	17-AUG-97	01-SEP-97	01-AUG-97

The ROUND and TRUNC functions can be used for number and date values. When used with dates, these functions round or truncate to the specified format model. Therefore, you can round dates to the nearest year or month.

Example

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and month started using the ROUND and TRUNC functions.

```
SELECT employee_id, hire_date, ROUND(hire_date, 'MONTH'), TRUNC(hire_date,
'MONTH')
FROM   employees
WHERE  hire_date LIKE '%97';
```

Conversion Functions

Data-type conversion
Implicit data-type
Explicit data-type

Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle9i database can be defined using ANSI, DB2, and SQL/DS data types. However, the Oracle Server internally converts such data types to Oracle8 data types. In some cases, the Oracle Server uses data of one data type where it expects data of a different data type.

This can happen when the Oracle Server can automatically convert the data to the expected data type. This data type conversion can be done implicitly by the Oracle Server, or explicitly by the user.Implicit data-type conversions work according to the rules explained in the next two slides. Explicit data-type conversions are done by using the conversion functions.

Note: Although implicit data-type conversion is available, it is recommended that you do explicit data type conversion to ensure the reliability of your SQL statements.

Implicit Data-Type Conversion

For assignments, the Oracle server can automatically convert the following:

From	To
VARCHAR2 or CHAR	**NUMBER**
VARCHAR2 or CHAR	**DATE**
NUMBER	**VARCHAR2**
DATE	**VARCHAR2**

Note: CHAR to NUMBER conversions succeed only if the character string represents a valid number.

Explicit Data-Type Conversion

TO_NUMBER
TO_DATE
TO_CHAR

SQL provides three functions to convert a value from one data type to another:
TO_CHAR(number|date,[fmt],[nlsparams]) Converts a number or date value to a VARCHAR2 character string with format model fmt.
Specifies the language in which month and day names and abbreviations are returned. If this parameter is omitted, this function uses the default
date languages for the session.
TO_NUMBER(char,[fmt], [nlsparams]) Converts a character string containing digits to a number in the format specified by the optional
format model fmt.
TO_DATE(char,[fmt],[nlsparams]) Converts a character string representing a date to a date value according to the fmt specified. If fmt is
omitted, the format is DD-MON-YY.

Using the TO_CHAR Function with Dates

TO_CHAR(date, 'format_model')

Guidelines

• The format model must be enclosed in single quotation marks and is case sensitive.
• The format model can include any valid date format element. Be sure to separate the date value from
the format model by a comma.
• The names of days and months in the output are automatically padded with blanks.
• To remove padded blanks or to suppress leading zeros, use the fill mode fm element.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

Elements of the Date Format Model

YYYY	**Full year in numbers**
YEAR	**Year spelled out**
MM	**Two-digit value for month**
MONTH	**Full name of the month**

MON	**Three-letter abbreviation of the month**
DY	**Day of the week**
DAY	**Full name of the day of the week**
DD	**Numeric day of the month**

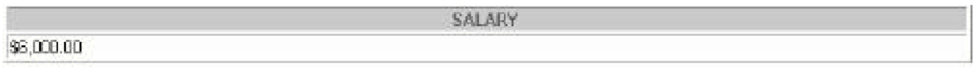
Sample Format Elements of Valid Date Formats

Element	
Q	Quarter of year

Using the TO_CHAR Function with Numbers

TO_CHAR(number, 'format_model')

When working with number values such as character strings, you s hould convert those numbers to the character data type using the TO_CHAR function, which translates a value of NUMBER data type to VARCHAR2 data type. This technique is especially useful with concatenation.



```
SALARY
$6,000.00
```

SELECT TO_CHAR(salary, '$99,999.00') SALARY FROM employees WHERE last_name = 'Ernst';

The TO_NUMBER and TO_DATE Functions

You may want to convert a character string to either a number or a date. To accomplish this task, you use the TO_NUMBER or TO_DATE functions. The format model you choose is based on the previously demonstrated format elements.

The fx modifier specifies exact matching for the character argument and date format model of a TO_DATE function:

• Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.

• The character argument cannot have extra blanks. Without fx, the Oracle Server ignores extra blanks.

• Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without fx, numbers in the character argument can omit leading zeroes.

Example

Display the names and hire dates of all the employees who joined on May 24, 1999. Because the fx modifier is used, an exact match is required and the spaces afte r the word "May" are not recognized.

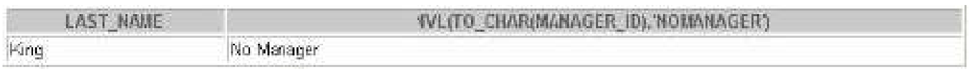
```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May       24, 1999', 'fxMonth DD, YYYY')
```

The RR Date Format Element

The RR date format is similar to the YY element, but you can use it to specify dif ferent centuries. You can use the RR date format element instead of YY, so that the centur y of the return value varies according to the specified two-digit year and the last two digits of the current year. The table on the slide summarizes the
behavior of the RR element.

Current Year	**Given Date**	**Interpreted (RR)**	**Interpreted (YY)**
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017

Nesting Functions
Single-row functions can be nested to any depth. Nested functions are evaluated from the innermost level to the outermost level. Some examples follow to show you the flexibility of these functions.



LAST_NAME	NVL(TO_CHAR(MANAGER_ID),'NOMANAGER')
King	No Manager

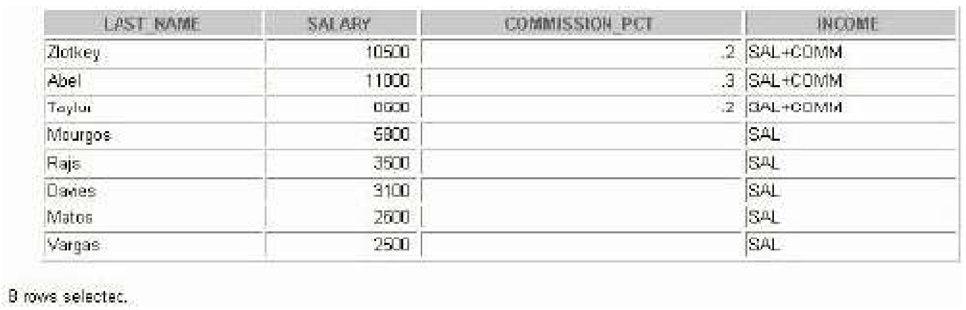
SELECT last_name, NVL(TO_CHAR(manager_id), 'No Manager') FROM employees WHERE manager_id IS NULL;
Nesting Functions (continued)

The slide example displays the head of the company, who has no manager. The evaluation of the SQL statement involves two steps:
1. Evaluate the inner function to convert a number value to a character string. - Result1 = TO_CHAR(manager_id)
2. Evaluate the outer function to replace the null value with a text string. - NVL(Result1, 'No Manager')
Example
Display the date of the next Friday that is six months from the hire date. The resulting date should appear as Friday, August 13th, 1999. Order the results by hire date.
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS (hire_date, 6), 'FRIDAY'), 'fmDay, Month DDth, YYYY') "Next 6 Month Review"
FROM employees ORDER BY hire_date;

General Functions
These functions work with any data type and pertain to the use of null values in the expression list.

Function Description
NVL Converts a null value to an actual value
NVL2 If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1can have any data type.
NULLIF Compares two expressions and returns null if they are equal, or the first expression if they are not equal
COALESCE Returns the first non-null expression in the expression list

The NVL Function
To convert a null value to an actual value, use the NVL function.
Syntax
NVL (expr1, expr2) In the syntax:
expr1 is the source value or expression that may contain a null expr2 is the target value for converting the null
You can use the NVL function to convert any data type, but the return value is always the same as the data type of expr1.



LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Zlotkey	10500	.2	SAL+COMM
Abel	11000	.3	SAL+COMM
Taylor	8600	.2	SAL+COMM
Mourgos	5800		SAL
Rajs	3500		SAL
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL

8 rows selected.

SELECT last_name, salary, commission_pct, NVL2(commission_pct, 'SAL+COMM', 'SAL') income FROM employees WHERE department_id IN (50, 80);

The NVL2 Function

The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned.

Syntax

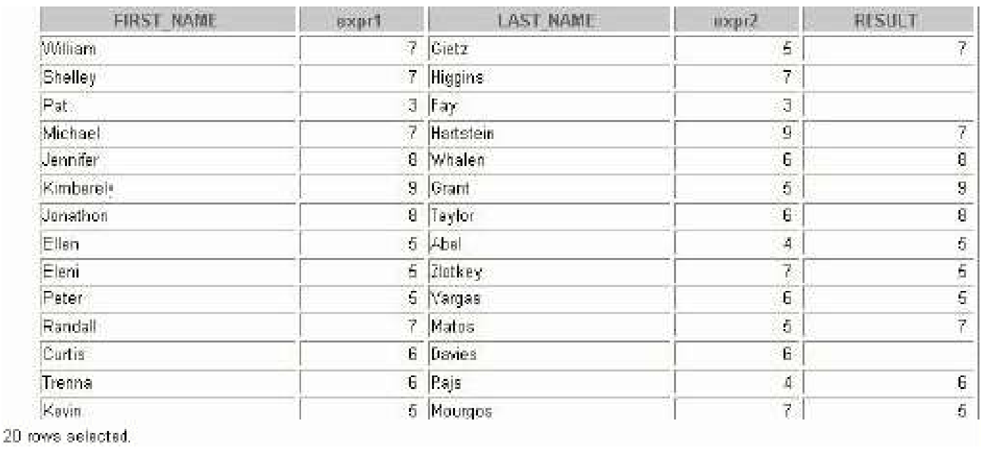
NVL(expr1, expr2, expr3) In the syntax:

expr1 is the source value or expression that may contain null expr2 is the value returned if expr1 is not null

expr3 is the value returned if expr2 is null

In the example shown, the COMMISSION_PCT column is examined. If a value is detected, the second expression of SAL+COMM is returned. If the COMMISSION_PCT column holds a null values, the third expression of SAL is returned. The argument expr1 can have any data type. The arguments expr2 and expr3 can have any data types except LONG. If the data types of expr2 and expr3 are different, The Oracle Server converts expr3 to the data type of expr2 before comparing them unless expr3 is a null constant. In that case, a data type

conversion is not necessary. The data type of the return value is always the same as the data type of expr2, unless expr2 is character data, in which case the return value's data type is VARCHAR2.



FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
William	7	Gietz	5	7
Shelley	7	Higgins	7	
Pat	3	Fay	3	
Michael	7	Hartstein	9	7
Jennifer	8	Whalen	6	8
Kimberely	9	Grant	5	9
Jonathon	8	Taylor	6	8
Ellen	5	Abel	4	5
Eleni	5	Zlotkey	7	5
Peter	5	Vargas	6	5
Randall	7	Matos	5	7
Curtis	6	Davies	6	
Trenna	6	Rajs	4	6
Kevin	5	Mourgos	7	5

20 rows selected.

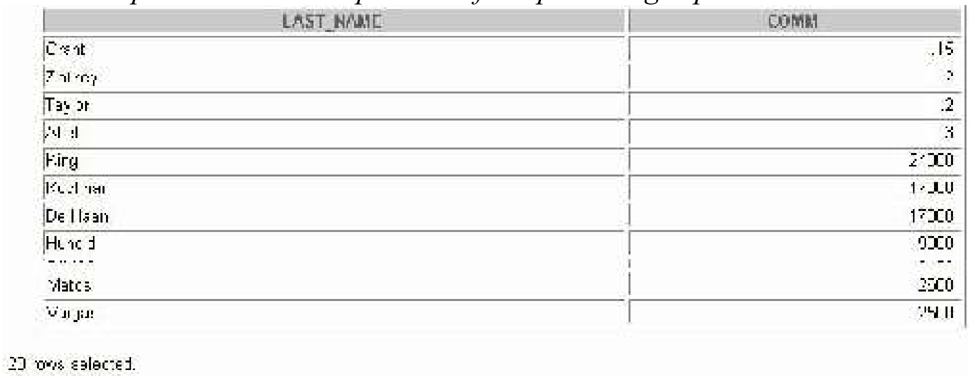
SELECT first_name, LENGTH(first_name) "expr1", last_name, LENGTH(last_name) "expr2", NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM employees;

The NULLIF Function

The NULLIF function compares two expressions. If they are equal, the function retur ns null. If they are not equal, the function returns the first expression. You cannot specify the literal NULL for first expression.

Syntax
NULLIF (expr1, expr2) In the syntax: *expr1 is the source value compared to expr2 expr2 is the source value compared with expr1. (If it is not equal to expr1, expr1* is returned.)

Using The COALESCE Function
The COALESCE function returns the first nonnull expression in the list.
Syntax
COALESCE (expr1, expr2, ... exprn) In the syntax: *expr1 returns this expression if it is not null expr2 returns this expression if the first expression is null and this expression is not* null *exprn returns this expression if the preceding expressions are null*



LAST_NAME	COMM
Grant	.15
Zlotkey	.2
Taylor	.2
Abel	.3
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Matos	2600
Vargas	2500

20 rows selected.

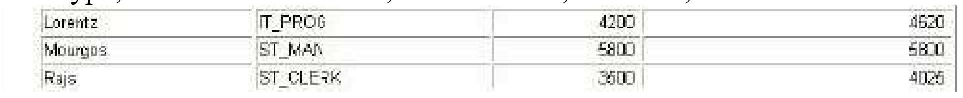
Using the COALESCE Function

SELECT last_name, COALESCE(commission_pct, salary, 10) comm FROM employees

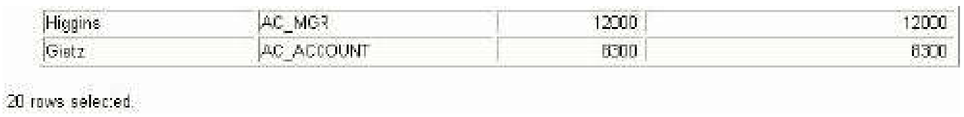
ORDER BY commission_pct;
In the example in the slide, if the COMMISSION_PCT value is not null, it is shown. If the COMMISSION_PCT value is null, then the SALARY is shown. If the COMMISSION_PCT and SALARY values are null, then the value 10 is shown.

Conditional Expressions
Two methods used to implement conditional processing (IF-THEN-ELSE logic) within a SQL statement are the CASE expression and the DECODE function.

The CASE Expression
CASE expressions let you use IF-THEN-ELSE logic in SQL statements without having to invoke procedures.
In a simple CASE expression, the Oracle Server searches for the first WHEN ... THEN pair for which expr is equal to comparison_expr and returns return_expr. If none of the WHEN ... THEN pairs meet this condition, and an ELSE clause exists, then Oracle returns else_expr. Otherwise, the Oracle Server returns null. You cannot specify the literal NULL for all the return_exprs and the else_expr.
All of the expressions (expr, comparison_expr, and return_expr) must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2.



Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025



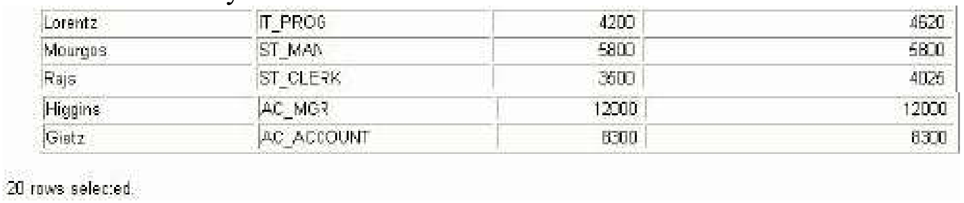
Higgins	AC_MGR	12000	12000
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

SELECT last_name, job_id, salary, CASE job_id WHEN 'IT_PROG' THEN 1.10*salary WHEN 'ST_CLERK' THEN 1.15*salary WHEN 'SA_REP' THEN 1.20*salary ELSE salary END "REVISED_SALARY" FROM employees;

In the preceding SQL statement, the value of JOB_ID is decoded. If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary. The same statement can be written with the DECODE function.

The DECODE Function
The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned. If the default value is omitted, a null value is returned where a search value does not match any of the result values.



Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
Higgins	AC_MGR	12000	12000
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

SELECT last_name, job_id, salary, DECODE(job_id, 'IT_PROG', 1.10*salary, 'ST_CLERK', 1.15*salary, 'SA_REP', 1.20*salary, salary)
REVISED_SALARY FROM employees;
Displaying Data from Multiple Tables

Objectives
After completing this lesson, you should be able to do the following:
• Write SELECT statements to access data from **more than one table using equality and nonequality joins**
• View data that generally does not meet a join **condition by using outer joins**
• Join a table to itself by using a self join

Obtaining Data from Multiple Tables
Sometimes you need to use data from more than one table.
• Employee IDs exist in the EMPLOYEES table.
• Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
• Location IDs exist in the DEPARTMENTS table.
To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables and access data from both of them.

Cartesian Products
When a join condition is invalid or omitted completely, the result is a Cartesian product, in which all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table. A Cartesian product tends to generate a large number of rows, and its result is rarely useful. You should always include a valid join condition in a WHERE clause, unless you have a specific need to combine all rows from all tables.
Cartesian products are useful for some tests when you need to ge nerate a large number of rows to simulate a reasonable amount of data.

Generating Cartesian Products

A Cartesian product is generated if a join condition is omitted. Display employee last name and department name from the EMPLOYEES and DEPARTMENTS tables without WHERE clause. All rows (20 rows) from the EMPLOYEES table are joined with all rows (8 rows) in the DEPARTMENTS table, thereby generating 160 rows in the output.
SELECT last_name, department_name dept_name FROM employees, departments;

Defining Joins

When data from more than one table in the database is required, a join condition is used. Rows in one table can be joined to rows in another table according to common values existing in corresponding columns, that is, usually primary and foreign key columns.
To display data from two or more related tables, write a simple join condition in the WHERE clause.
In the syntax: *table1.column denotes the table and column from which data is retrieved table1.column1 = table2.column2 is the condition that joins (or relates) the tables together*

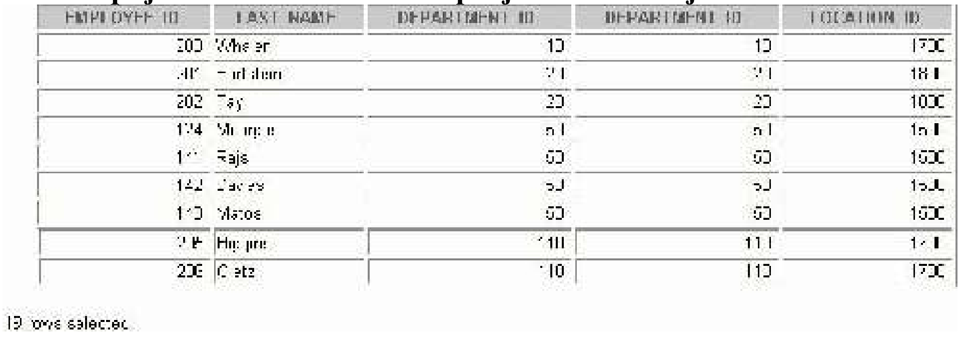
Guidelines

• When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
• If the same column name appears in more than one table, the column name must be prefixed with the table name.
• To join n tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.

Equijoins

To determine an employee's department name, you compare the value in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an equijoin, that is, values in the DEPARTMENT_ID column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

Note: Equijoins are also called simple joins or inner joins.



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
205	Higgins	110	110	1700
206	Gietz	110	110	1700

19 rows selected.

SELECT employees.employee_id, employees.last_name, employees.department_id, departments.department_id, departments.location_id
FROM employees, departments WHERE employees.department_id = departments.department_id;

In the slide:
• The SELECT clause specifies the column names to retrieve:
- Employee last name, employee number, and department number, which are columns in the EMPLOYEES table
- Department number, department name, and location ID, which are columns in the DEPARTMENTS table
• The FROM clause specifies the two tables that the database must access:
- EMPLOYEES table
- DEPARTMENTS table
• The WHERE clause specifies how the tables are to be joined:
EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
Because the DEPARTMENT_ID column is common to both tables, it must be prefixed by the table name to avoid ambiguity.

Additional Search Conditions
In addition to the join, you may have criteria for your WHERE clause to restrict the rows under consideration for one or more tables in the join. For example, to display employee Matos' department number and department name, you need an additional condition in the WHERE clause.

```
SELECT last_name, employees.department_id, department_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id
AND    last_name = 'Matos';
```

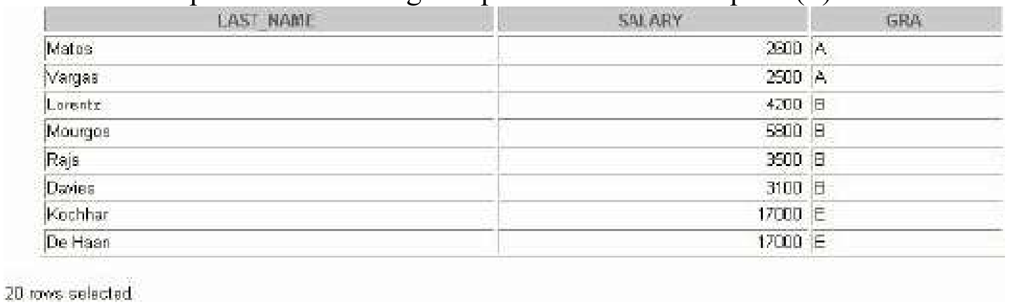
Qualifying Ambiguous Column Names
You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT_ID column could be from either the DEPARTMENTS table or the EMPLOYEES table. It is necessary to add the table prefix to execute your query. If there are no common column names between the two tables, there is no need to qualify the columns. **However, using the table prefix improves performance, because you tell the Oracle Server exactly where to find the columns.**

Table Aliases
Qualifying column names with table names can be very time consum ing, particularly if table names are lengthy. You can use table aliases instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. **Table aliases help to keep SQL code smaller, therefore using less memory.**The table name is specified in full in the From clause, followed by a space and then the table alias.

Guidelines
• Table aliases can be up to 30 characters in length, but the shorter they are the better.
• If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
• Table aliases should be meaningful.
• The table alias is valid only for the current SELECT statement.
Additional Search Conditions
Sometimes you may need to join more than two tables. For example, to display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

```
SELECT e.last_name, d.department_name, l.city FROM   employees e, departments d,
locations l WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

Nonequijoins
A nonequijoin is a join condition containing something other than an equality operator. The relationship between the EMPLOYEES table and the JOB_GRADES table has an example of a nonequijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST_SALARY and HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equals (=).



LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Kochhar	17000	E
De Haan	17000	E

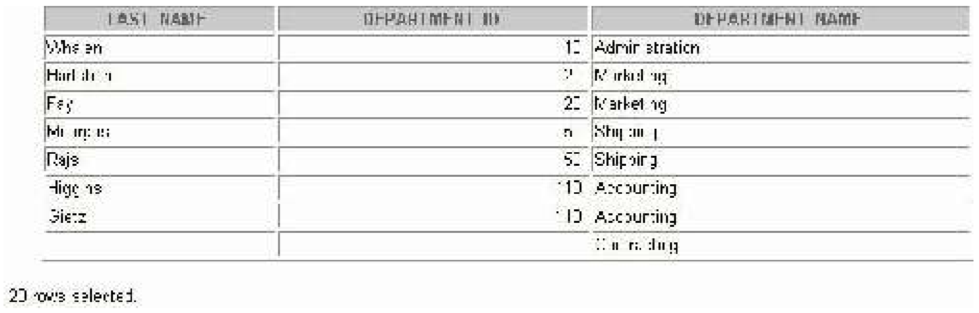
20 rows selected

SELECT e.last_name, e.salary, j.grade_level FROM employees e, job_grades j
WHERE e.salary BETWEEN j.lowest_sal AND j.highest _sal;
Retrieving Records with Nonequijoins

The example in the slide creates a nonequijoin to evaluate an employee's salary grade. The salary must be *between any pair of the low and high salary ranges.*
It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:
• None of the rows in the job grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary val ues of one of the rows in the salary grade table.
• All of the employees' salaries lie within the limits provided by the job grade table. That is, no employee earns less than the lowest value contained in the LOWEST_SAL column or more than the highest value contained in the HIGHEST_SAL column.
Note: Other conditions, such as <= and >= can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using BETWEEN. Table aliases have been specified in the slide example for performance reasons, not because of possible
ambiguity.

Returning Records with No Direct Match with Outer Joins
If a row does not satisfy a join condition, the row will not appear in the query result. For example, in the equijoin condition of EMPLOYEES and DEPARTMENTS tables, an employee Grant may not appear if there is no department ID recorded.

Using Outer Joins to Return Records with No Direct Match
The missing rows can be returned if an outer join operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the side of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more rows from the nondeficient table can be joined. In the syntax:
table1.column = is the condition that joins (or relates) the tables together.
table2.column (+) is the outer join symbol, which can be placed on either side of the WHERE clause condition, but not on both sides. (Place the outer
join symbol following the name of the column in the table without the matching rows.)



LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Higgins	110	Accounting
Gietz	110	Accounting
		Contracting

20 rows selected.

SELECT e.last_name, e.department_id, d.department_name FROM employees e, departments d WHERE e.department_id(+) = d.department_id;

Outer Join Restrictions

• The outer join operator can appear on only one side of the expression: the side that has information missing. It returns those rows from one table that have no direct match in the other table.

• A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator.

Joining a Table to Itself

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self join.

SELECT worker.last_name || ' works for ' || manager.last_name FROM employees worker, employees manager WHERE worker.manager_id = manager.employee_id;

<u>Aggregating Data Using Group Functions</u>

Objectives

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

• Identify the available group functions
• Describe the use of group functions
• Group data using the GROUP BY clause
• Include or exclude grouped rows by using the **HAVING clause**

Group Functions

Unlike single-row functions, group functions operate on sets of rows to give one result per group. These sets may be the whole table or the table split into groups.

Types of Group Functions

Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

Function	**Description**
AVG([DISTINCT|ALL]n)	Average value of n, ignoring null values
COUNT({*|[DISTINCT| ALL]expr})	Number of rows, where expr evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX([DISTINCT|ALL]expr)	Maximum value of expr, ignoring null values
MIN([DISTINCT|ALL]expr)	Minimum value of expr, ignoring null values

STDDEV([DISTINCT|ALL]x)
Standard deviation of n, ignoring null values
SUM([DISTINCT|ALL]n)
Sum values of n, ignoring null values
VARIANCE([DISTINCT| ALL]x)
Variance of n, ignoring null values

Guidelines for Using Group Functions

• DISTINCT makes the function consider only nonduplicate values; ALL makes it consider every value including duplicates. The default is ALL and therefore does not need to be specified.
• The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.
• All group functions ignore null values. To substitute a value for null values, use the NVL, NVL2, or COALESCE functions.
• The Oracle Server implicitly sorts the result set in ascending order when using a GROUP BY clause. To override this default ordering, DESC can be used in an ORDER BY clause.

You can use AVG and SUM for numeric data.

**SELECT AVG(salary), MAX(salary), MIN(salary), SUM(salary)
FROM employees WHERE job_id LIKE '%REP%';
Group Functions
You can use AVG, SUM, MIN, and MAX functions against columns that can store numeric data. The example displays the average, highest, lowest, and sum of monthly salaries for all sales representatives.
SELECT MIN(hire_date), MAX(hire_date) FROM employees;
Min and Max Function
You can use the MIN and MAX functions for any data type. The example displays the most junior and most senior employee.
Note: AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types.
COUNT(*) returns the number of rows in a table.
SELECT COUNT(*) FROM employees WHERE department_id = 50;
The COUNT Function
The COUNT function has three formats:**
- **COUNT(*)**
- **COUNT(expr)**
- **COUNT(DISTINCT expr)**

**COUNT(*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns. If a WHERE clause is included in the SELECT statement, COUNT(*) returns the number of rows that satisfies the condition in the WHERE clause.
In contrast, COUNT(expr) returns the number of nonnull values in the column identified by expr.
COUNT(DISTINCT expr) returns the number of unique, non-null values in the column identified by *expr.***
- **COUNT(expr) returns the number of rows with non-null values for the expr.**

**SELECT COUNT(commission_pct) FROM employees WHERE department_id = 80;
Using the DISTINCT Keyword**

• COUNT(DISTINCT expr) returns the number of distinct nonnull values of the expr.

```
SELECT COUNT(DISTINCT department_id) FROM   employees;
SELECT AVG(commission_pct) FROM   employees;
```

Group Functions and Null Values

All group functions ignore null values in the column. In the example, the average is calculated based only on the rows in the table where a valid value is stored in the COMMISSION_PCT column. The average is calculated as the total commission paid to all employees divided by the number of employees receiving a commission.

```
SELECT AVG(NVL(commission_pct, 0)) FROM   employees;
```

The NVL function forces group functions to include null values. In the example, the average is calculated based on all rows in the table, regardless of whether null values are stored in the COMMISSION_PCT column. The average is calculated as the total commission paid to all employees divided by the total number of employees in the company.

Creating Groups of Data:

Divide rows in a table into smaller groups by using the GROUP BY clause.

The GROUP BY Clause

You can use the GROUP BY clause to divide the rows in a table into groups. You can then use the group functions to return summary information for each group.

In the syntax:

***group_by_expression specifies columns whose values determine the basis for* grouping rows**

Guidelines

• If you include a group function in a SELECT clause, you cannot select individual results as well, *unless the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.*

• Using a WHERE clause, you can exclude rows before dividing them into groups.

• You must include the columns in the GROUP BY clause.

• You cannot use a column alias in the GROUP BY clause.

• By default, rows are sorted by ascending order of the columns included in the GROUP BY list. You can override this by using the ORDER BY clause.

```
SELECT department_id, AVG(salary) FROM   employees
GROUP BY department_id;
```

Here is how this SELECT statement, containing a GROUP BY clause, is evaluated:

• The SELECT clause specifies the columns to be retrieved:

- The department number column in the EMPLOYEES table

- The average of all the salaries in the group you specified in the GROUP BY clause

• The FROM clause specifies the tables that the database must access: the EMPLOYEES table.

• The WHERE clause specifies the rows to be retrieved. Because there is no WHERE clause, all rows are retrieved by default.

• The GROUP BY clause specifies how the rows should be grouped. The rows are being grouped by department number, so the AVG function that is being applied to the salary column will calculate the average salary for each department.

```
SELECT   AVG(salary) FROM     employees
GROUP BY department_id;
```

The GROUP BY column does not have to be in the SELECT clause. For example, the SELECT statement displays the average salaries for each department without displaying the respective department numbers. Without the department numbers, however, the results do not look meaningful.

You can use the group function in the ORDER BY clause.

```
SELECT   department_id, AVG(salary) FROM     employees
```

GROUP BY department_id ORDER BY AVG(salary);

Grouping by More Than One Column

Groups within Groups
Sometimes you need to see results for groups within groups. The slide shows a report that displays the total salary being paid to each job title, within each department.
The EMPLOYEES table is grouped first by department number and, within that grouping, by job title.

```
SELECT   department_id dept_id, job_id, SUM(salary) FROM     employees
GROUP BY department_id, job_id;
```

Using the Group By Clause on Multiple Columns
You can return summary results for groups and subgroups by listing more than one GROUP BY column. You can determine the default sort order of the results by the order of the columns in the GROUP BY clause. Here is how the SELECT statement on the slide, containing a GROUP BY clause, is evaluated:
• The SELECT clause specifies the column to be retrieved:
- Department number in the EMPLOYEES table
- Job ID in the EMPLOYEES table
- The sum of all the salaries in the group that you specified in the GROUP BY clause
• The FROM clause specifies the tables that the database must access: the EMPLOYEES table
• The GROUP BY clause specifies how you must group the rows:
- First, the rows are grouped by department number
- Second, within the department number groups, the rows are grouped by job ID
So the SUM function is being applied to the salary column for all job IDs within each department number group.

• You cannot use the WHERE clause to **restrict groups.**
• You use the HAVING clause to restrict groups.
• You cannot use group functions in the WHERE **clause.**

use the HAVING clause to restrict groups.

```
SELECT department_id, AVG(salary) FROM   employees
GROUP BY department_id
HAVING  AVG(salary) > 8000 ;
```

The Oracle Server performs the following steps when you use the HAVING clause:
1. Rows are grouped.
2. The group function is applied to the group.
3. The groups that match the criteria in the HAVING clause are displayed.
The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because that is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

SELECT job_id, SUM(salary) PAYROLL FROM employees
WHERE job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING SUM(salary) > 13000
ORDER BY SUM(salary);
The example isplays the job ID and total monthly salary for each job with a total payroll exceeding $13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

Nesting Group Functions

Display the maximum average salary.
SELECT MAX(AVG(salary)) FROM employees
GROUP BY department_id;
Group functions can be nested to a depth of two. The example in the slide displays the maximum average salary.

Subqueries
Objectives

After completing this lesson, you should be able to do the following:
• Describe the types of problem that subqueries can **solve**
• Define subqueries
• List the types of subqueries
• Write single-row and multiple-row subqueries

Using a Subquery to Solve a Problem
Suppose you want to write a query to find out who earns a salary greater than Abel's salary. To solve this problem, you need two queries: one to find what Abel earns, and a second query to find who earns more than that amount.
You can solve this problem by combining the two queries, placing one query inside the other query. The inner query, also called the subquery, returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

Subqueries
A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.
You can place the subquery in a number of SQL clauses, including:
• The WHERE clause
• The HAVING clause
• The FROM clause
In the syntax:
operator includes a comparison condition such as >, =, or IN
Note: Comparison conditions fall into two classes: single -row operators (>, =, >=, <, <>, <=) and multiple- row operators (IN, ANY, ALL).
The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement. The subquery generally executes first, and its output is used to

complete the query condition for the main or outer query.

```
SELECT last_name FROM   employees WHERE  salary >
(SELECT salary FROM employees WHERE  last_name = 'Abel');
```

the inner query determines the salary of employee Abel. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than this amount.

Guidelines for Using Subqueries

- A subquery must be enclosed in parentheses.
- Place the subquery on the right side of the comparison condition for readability.
- Starting with release Oracle8i, an ORDER BY clause can be used and is required in the subquery to perform top-n analysis.
- Two classes of comparison conditions are used in subqueries: single-row operators and multiple-row operators.

Types of Subqueries

- Single-row subqueries: Queries that return only one row from the inner SELECT statement
- Multiple-row subqueries: Queries that return more than one row from the inner SELECT statement

Note: There are also multiple-column subqueries: queries that return more than one column from the inner SELECT statement.

Single-Row Subqueries

A single-row subquery is one that returns one row from the inner SELECT statement. This type of subquery uses a single -row operator.

Example

Display the employees whose job ID is the same as that of employee 141.

```
SELECT last_name, job_id FROM   employees WHERE  job_id =
 (SELECT job_id FROM   employees WHERE  employee_id = 141);
```

Executing Single-Row Subqueries

A SELECT statement can be considered as a query block. The example displays employees whose job ID is the same as that of employee 141 The example consists of two query blocks: the outer query and two inner queries. The inner query blocks are executed first. The outer query block is then processed and uses the values returned by the inner query to complete its search conditions.

Note: The outer and inner queries can get data from different tables.

```
SELECT last_name, job_id, salary FROM   employees WHERE  salary =
(SELECT MIN(salary) FROM   employees);
```

Using Group Functions in a Subquery

You can display data from a main query by using a group function in a subquery to return a single row. The
subquery is in parentheses and is placed after the comparison condition.

```
SELECT   department_id, MIN(salary) FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >  (SELECT MIN(salary) FROM employees WHERE
department_id = 50);
```

The HAVING Clause with Subqueries

You can use subqueries not only in the WHERE clause, but also in the HAVING clause. The Oracle Server executes the subquery, and the results are returned into the HAVING clause of the main query. The SQL statement on the slide displays all the departments that have a minimum salary greater than that of department 50.

Multiple-Row Subqueries
Subqueries that return more than one row are called multiple-row subqueries. You use a multiple-row operator, instead of a single-row operator, with a multiple -row subquery. The multiple-row operator expects one or more values.
SELECT last_name, salary, department_id FROM employees WHERE salary IN (SELECT MIN(salary) FROM employees GROUP BY department_id);
Example
Find the employees who earn the same salary as the minimum salar y for each department.
The inner query is executed first, producing a query result. The main query block is then processed and uses the values returned by the inner query to complete its search condition. In fact, the main query would look like the following to the Oracle Server:
SELECT last_name, salary, department_id FROM employees
WHERE salary IN (2500, 4200, 4400, 6000, 7000, 8300, 8600, 17000);

SELECT employee_id, last_name, job_id, salary FROM employees WHERE salary < ANY
(SELECT salary FROM employees WHERE job_id = 'IT_PROG') AND job_id <> 'IT_PROG';

The ANY operator (and its synonym the SOME operator) compares a value to each value returned by a subquery. The example displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is $9,000. <ANY means less than the maximum. >ANY means more than the minimum. =ANY is equivalent to IN.

SELECT employee_id, last_name, job_id, salary FROM employees
WHERE salary < ALL (SELECT salary FROM employees WHERE job_id = 'IT_PROG') AND job_id <> 'IT_PROG';

The ALL operator compares a value to every value returned by a subquery. The example displays employees whose salary is less than the salary of all employees with a job ID of IT_PROG and whose job is not IT_PROG. >ALL means more than the maximum, and <ALL means less than the minimum.
The NOT operator can be used with IN, ANY, and ALL operators.

Null Values in a Subquery

SELECT emp.last_name FROM employees emp WHERE emp.employee_id NOT IN
(SELECT mgr.manager_id FROM employees mgr);
Returning Nulls in the Resulting Set of a Subquery

The SQL statement in the slide attempts to display all the employees who do not have any subordinates. Logically, this SQL statement should have returned 12 rows. Howe ver, the

SQL statement does not return any rows. One of the values returned by the inner query is a null value, and hence the entire query returns no rows. The reason is that all conditions that compare a null value result in a null. So whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator. The NOT IN operator is equivalent to <> ALL.

Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator.

The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use
the following SQL statement:
SELECT emp.last_name FROM   employees emp
WHERE emp.employee_id IN (SELECT mgr.manager_id FROM   employees mgr);
Alternatively, a WHERE clause can be included in the subquery to display all the employees who do not have any subordinates:
SELECT last_name FROM   employees WHERE employee_id NOT IN
(SELECT manager_id FROM   employees WHERE manager_id IS NOT NULL);

Manipulating Data

Objectives

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

- Describe each DML statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Merge rows in a table
- Control transactions

Data Manipulation Language
Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a transaction.
Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decrease the savings account, increase the checking account, and record the transaction in the transaction journal. The Oracle Server must guarantee that all three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

Adding a New Row to a Table
You can add new rows to a table by issuing the INSERT statement. In the syntax:
table is the name of the table column is the name of the column in the table to populate value is the corresponding value for the column
Note: This statement with the VALUES clause adds only one row at a time to a table.

INSERT INTO departments(department_id, department_name, manager_id, location_id)
VALUES   (70, 'Public Relations', 100, 1700);
1 row created.

• Enclose character and date values within single **quotation marks.**
Because you can insert a new row that contains values for each column, the column list is not required in the INSERT clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.
For clarity, use the column list in the INSERT clause.
Enclose character and date values within single quotation marks; it is not recommended to enclose numeric values within single quotation marks.
Number values should not be enclosed in single quotes, because implicit conversion may take place for numeric values assigned to NUMBER data type columns if single quotes are included.

Methods for Inserting Null Values
Method Description
Implicit Omit the column from the column list.
Explicit Specify the NULL keyword in the VALUES list, specify the empty string ('') in the VALUES list for character strings and
dates.

INSERT INTO employees (employee_id, first_name, last_name, email, phone_number,hire_date, job_id, salary,
commission_pct, manager_id, department_id)
VALUES (113, 'Louis', 'Popp', 'LPOPP', '515.124.4567', SYSDATE, 'AC_ACCOUNT', 6900, NULL, 205, 100);
1 row created.

Inserting Special Values by Using SQL Functions
You can use functions to enter special values in your table. It supplies the current date and time in the HIRE_DATE column. It uses the SYSDATE function for current date and time. You can also use the USER function when inserting rows in a table. The USER function records the current username.

Inserting Specific Date and Time Values
The DD-MON-YY format is usually used to insert a date value. With this format, recall that the century defaults to the current century. Because the date also contains time information, the default time is midnight (00:00:00).if a date must be entered in a format other than the default format (for example, with another century, or
a specific time), you must use the TO_DATE function.

INSERT INTO sales_reps(id, name, salary, commission_pct) SELECT employee_id, last_name, salary, commission_pct FROM employees
WHERE job_id LIKE '%REP%';
4 rows created.

Copying Rows from Another Table
You can use the INSERT statement to add rows to a table where the values are derived from existing tables. In place of the VALUES clause, you use a subquery.

Changing Data in a Table

Updating Rows

You can modify existing rows by using the UPDATE statement. In the syntax:

table is the name of the table

column is the name of the column in the table to populate

value is the corresponding value or subquery for the column

condition identifies the rows to be updated and is composed of column names expressions, constants, subqueries, and comparison operators

Note: In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

UPDATE employees SET department_id = 70 WHERE employee_id = 113;
1 row updated.

All rows in the table are modified if you omit the **WHERE clause.**

Update employee 114's job and department to match that of employee 205.

UPDATE employees SET job_id = (SELECT job_id FROM employees WHERE employee_id = 205),
salary = (SELECT salary FROM employees WHERE employee_id = 205) WHERE employee_id = 114;
1 row updated.

Updating Two Columns with a Subquery

You can update multiple columns in the SET clause of an UPDATE statement by writing multiple subqueries.

Use subqueries in UPDATE statements to update rows in a table based on values from another table.

UPDATE copy_emp SET department_id = (SELECT department_id FROM employees
WHERE employee_id = 100) WHERE job_id = (SELECT job_id FROM employees
WHERE employee_id = 200);
1 row updated.

Removing a Row from a Table

The DELETE Statement

You can remove existing rows from a table by using the DELETE statement.

Deleting Rows

You can remove existing rows by using the DELETE statement. In the syntax:

table is the table name

condition identifies the rows to be deleted and is composed of column names, expressions, constants, subqueries, and comparison operators

DELETE FROM departments WHERE department_name = 'Finance';
1 row deleted.

• All rows in the table are deleted if you omit the **WHERE clause.**

Use subqueries in DELETE statements to remove rows from a table based on values from another table.

DELETE FROM employees WHERE department_id = (SELECT department_id FROM departments WHERE department_name LIKE '%Public%');

1 row deleted.

Deleting Rows Based on Another Table

You can use subqueries to delete rows from a table based on values from another table. The example deletes all the employees who are in a department where the department name contains the Public string. The subquery searches the DEPARTMENTS table to find the department number based on the department name containing the Public string. The subquery then feeds the department number to the main query, which deletes rows of data from the EMPLOYEES table based on this department number.

INSERT INTO (SELECT employee_id, last_name, email, hire_date, job_id, salary, department_id
FROM employees WHERE department_id = 50) VALUES (99999, 'Taylor', 'DTAYLOR', TO_DATE('07-JUN-99', 'DD-MON-RR'),
'ST_CLERK', 5000, 50);
1 row created.

Using a Subquery in an INSERT Statement

You can use a subquery in place of the table name in the INTO clause of the INSERT statement. The select list of this subquery must have the same number of columns as the column list of the VALUES clause. Any rules on the columns of the base table must be follo wed in order for the INSERT statement to work successfully. For example, you could not put in a duplicate employee ID, nor leave out a value for a mandatory not null column.

MERGE Statements

SQL has been extended to include the MERGE statement. Using this statement, you can update or insert a row conditionally into a table, thus avoiding multiple UPDATE statements. The decision whether to update or insert into the target table is based on a condition in the ON clause. Because the MERGE command combines the INSERT and UPDATE commands, you need both INSERT and UPDATE privileges on the target table and the SELECT privilege on the source table.The MERGE statement is deterministic. You cannot update the same row of the target table multiple times in the same MERGE statement.An alternative approach is to use PL/SQL loops and multiple DML statements. The MERGE statement, however, is easy to use and more simply expressed as a single SQ L statement.
The MERGE statement is suitable in a number of data warehousing applications. For example, in a data warehousing application, you may need to work with data coming from multiple sources, some of which may be duplicates. With the MERGE statement, you can conditionally add or modify rows.

<u>MERGE Statement Syntax</u>

MERGE INTO table_name AS table_alias USING (table|view|sub_query) AS alias ON (join condition)
WHEN MATCHED THEN
UPDATE SET ***col1 = col_val1, col2 = col2_val***
WHEN NOT MATCHED THEN
INSERT (column_list) VALUES (column_values);
MERGE INTO copy_emp AS c USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name, c.last_name = e.last_name,

```
...
c.department_id  = e.department_id
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name, e.email,
e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id, e.department_id);
```

Database Transactions

The Oracle Server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure. Transactions consist of DML statements that make up one consistent change to the data. For example, a transfer of funds between two accounts should include the debit to one account and the credit to another account in the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

When Does a Transaction Start and End?

A transaction begins when the first DML statement is encountered and ends when one o f the following occurs:

• A COMMIT or ROLLBACK statement is issued
• A DDL statement, such as CREATE, is issued
• A DCL statement is issued
• The user exits iSQL*Plus
• A machine fails or the system crashes

After one transaction ends, the next executable SQL statement automatically starts the next transaction.
A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

<u>Controlling Transactions</u>

Explicit Transaction Control Statements

You can control the logic of transactions by using the COMMIT, SAVEPOINT, and ROLLBACK statements.

Statement	**Description**
COMMIT	Ends the current transaction by making all pending data changes permanent
SAVEPOINT name	Marks a savepoint within the current transaction
ROLLBACK	ROLLBACK ends the current transaction by discarding all pending data changes
ROLLBACK TO SAVEPOINT	rolls back the current transaction to the specified savepoint, thereby discarding any changes and or savepoints created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. As savepoints are logical, there is no way to list the savepoints you have created.

Rolling Back Changes to a Savepoint

You can create a marker in the current transaction by using the SAVEPOINT statement which divides the transaction into smaller sections. You can then discard pending changes up to that marker by using the ROLLBACK TO SAVEPOINT statement. If you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

Implicit Transaction Processing

Status	**Circumstances**
Automatic commit	DDL statement or DCL statement is issued.
	*iSQL*Plus exited normally, without explicitly issuing COMMIT or* ROLLBACK commands.
Automatic rollback	Abnormal termination of iSQL*Plus or system failure.

System Failures
When a transaction is interrupted by a system failure , the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to their state at the time of the last commit. In this way, the Oracle Server protects the integrity of the tables.

Committing Changes
Every data change made during the transaction is temporary until the transaction is committed. State of the data before COMMIT or ROLLBACK statements are issued:
• Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
• The current user can review the results of the data manipulation operations by querying the tables.
• Other users cannot view the results of the data manipulation ope rations made by the current user. The Oracle Server institutes read consistency to ensure that each user sees data as it existed at the last commit.
• The affected rows are locked; other users cannot change the data in the affected rows.
Make all pending changes permanent by using the COMMIT statement. Following a COMMIT statement:
• Data changes are written to the database.
• The previous state of the data is permanently lost.
• All users can view the results of the transaction.
• The locks on the affected rows are released; the rows are now available for other users to perform new data changes.
• All savepoints are erased.

Statement-Level Rollback

• If a single DML statement fails during execution, **only that statement is rolled back.**
• The Oracle Server implements an implicit **savepoint.**
• All other changes are retained.
• The user should terminate transactions explicitly **by executing a COMMIT or ROLLBACK statement.**

Statement-Level Rollbacks
Part of a transaction can be discarded by an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of a transaction, its effect is undone by a statement-level rollback, but the changes made by the previous DML statements in the tra nsaction are not discarded. They
can be committed or rolled back explicitly by the user.The Oracle Server issues an implicit commit before and after any data definition language (DDL) statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit. Terminate your transactions explicitly by executing a COMMIT or ROLLBACK statement.

Read Consistency

Database users access the database in two ways:
• Read operations (SELECT statement)
• Write operations (INSERT, UPDATE, and DELETE statements)
You need read consistency so that the following occur:
• The database reader and writer are ensured a consistent view of the data.
• Readers do not view data that is in the process of being changed.
• Writers are ensured that the changes to the database are done in a consistent way.
• Changes made by one writer do not disrupt or conflict with changes another writer is making.
The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

Locking

What Are Locks?
Locks are mechanisms that prevent destructive interaction between tra nsactions accessing the same resource, either a user object (such as tables or rows) or a system object not visible to users (such as shared data structures and data dictionary rows).

How the Oracle Database Locks Data
Locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Implicit locking occurs for all SQL statements except SELECT.
The users can also lock data manually, which is called explicit locking.

DML Locking
When performing data manipulation language (DML) operations, the Oracle Server provides data concurrency through DML locking. DML locks occur at two levels:
• A share lock is automatically obtained at the table level during DML operations. With share lock mode, several transactions can acquire share locks on the same resource.
• An exclusive lock is acquired automatically for each row modified by a DML statement. Exclusive locks prevent the row from being changed by other transactions until the transaction is committed or rolled back. This lock ensures that no other user can modify the same row at the same time and overwrite changes not yet committed by another user.
Note: DDL locks occur when you modify a database object such as a table.

Creating and Managing Tables

Objectives

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

• Describe the main database objects
• Create tables
• Describe the data types that can be used when **specifying column definition**
• Alter table definitions
• Drop, rename, and truncate tables

Database Objects

Table Basic unit of storage; composed of rows and columns
View Logically represents subsets of data from one or more tables
Sequence Numeric value generator
Index Improves the performance of some queries
Synonym Gives alternative names to objects
Oracle9i Table Structures

• Tables can be created at any time, even while users are using the database.
• You do not need to specify the size of any table. The size is ultimately defined by the amount of space allocated to the database as a whole. It is important, however, to estimate how much space a table will use over time.
• Table structure can be modified online.

Naming Rules
Name database tables and columns according to the standard rules for naming any Oracle database object:
• Table names and column names must begin with a letter and be 1 -30 characters long.
• Names must contain only the characters A-Z, a-z, 0-9, _ (underscore), $, and # (legal characters, but their use is discouraged).
• Names must not duplicate the name of another object owned by the same Oracle Server user.
• Names must not be an Oracle Server reserved word.

Naming Guidelines
Use descriptive names for tables and other database objects.
Note: Names are case insensitive. For example, EMPLOYEES is treated as the same name as eMPloyees or eMpLOYEES.

The CREATE TABLE Statement
Create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the data definition language (DDL) statements, which are covered in subsequent lessons. DDL statements are a subset of SQL statements used to create, modify, or remove Oracle9i database structures. These statements
have an immediate effect on the database, and they also record information in the data dictionary. To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator uses data control language (DCL) statements, which are covered in a
later lesson, to grant privileges to users.
In the syntax: *schema is the same as the owner's name*
table is the name of the table
DEFAULT expr specifies a default value if a value is omitted in the INSERT statement
column is the name of the column datatype is the column's data type and length

Referencing Another User's Tables
A schema is a collection of objects. Schema objects are the logical structures that directly refer to the data in a database. Schema objects include tables, views, synonyms, sequen ces, stored procedures, indexes, clusters, and database links. If a table does not belong to the user, the owner's name must be prefixed to the table. For example, if there is a schema named USER_B, and USER_B has an EMPLOYEES table, then specify the

following to retrieve data from that table:
SELECT * FROM user_b.employees;

The DEFAULT Option
A column can be given a default value by using the DEFAULT option. This option prevents null values from entering the columns if a row is inserted without a value for the column. The default value can be a literal value, an expression, or a SQL function, such as SYSDATE and USER, but the value cannot be the name of another column or a pseudocolumn, such as NEXTVAL or CURRVAL. The default expression must match the data type of the column.

CREATE TABLE dept (deptno NUMBER(2),dname VARCHAR2(14), loc VARCHAR2(13));
Table created.
• Confirm creation of the table.
DESCRIBE dept

Creating Tables
The example on the slide creates the DEPT table, with three columns: namely, DEPTNO, DNAME, and LOC. It further confirms the creation of the table by issuing the DESCRIBE command. Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.
Introduction to Oracle9i: SQL 9-8

<u>Tables in the Oracle Database</u>

• User tables:
- Are a collection of tables created and maintained by
the user
- Contain user information
• Data dictionary:
- Is a collection of tables created and maintained by

the Oracle Server
- Contain database information

Tables in the Oracle Database
User tables are tables created by the user, such as EMPLOYEES. There is another collection of tables and views in the Oracle Database known as the data dictionary. This collection is created and maintained by the Oracle Server and contains information about the database. All data dictionary tables are owned by the SYS user. The base tables are rarely accessed by the user because the information in them is not easy to understand. Therefore, users typically access data dictionary views because the information is presented in a format that is easier to understand. Information stored in the data dictionary includes names of the Oracle Server users, privileges granted to users, database object names, table constraints, and auditing information.There are four categories of data dictionary views; each category has a distinct prefix that reflects its intended use.

Prefix	**Description**
USER_	These views contain information about objects owned by the user .
ALL_	These views contain information about all of the tables (object tables and relational tables) accessible to the user.
DBA_	These views are restricted views, which can be accessed only by people who have been assigned the DBA role.
V$	These views are dynamic performance views, database server performance, memory, and locking.

Data Types

Data Type	**Description**
VARCHAR2(size)	**Variable-length character data**
CHAR[(size)]	**Fixed-length character data**
NUMBER[(p,s)]	**Variable-length numeric data**
DATE	**Date and time values**
LONG	**Variable-length character data up to 2 gigabytes**
CLOB	**Character data up to 4 gigabytes**
RAW and LONG RAW	**Raw binary data**
BLOB	**Binary data up to 4 gigabytes**
BFILE	**Binary data stored in an external file; up to 4 gigabytes**
ROWID	**Hexadecimal string representing the unique address of a row in its table**

The ALTER TABLE Statement
After you create a table, you may need to change the table structure because you omitted a column or your column definition needs to be changed or you need to remove columns. You can do this by using the ALTER TABLE statement.
You can add, modify, and drop columns to a table by using the ALTER TABLE statement.
In the syntax:
table is the name of the table
ADD|MODIFY|DROP is the type of modification
column is the name of the new column
datatype is the data type and length of the new column
DEFAULT expr specifies the default value for a new column

ALTER TABLE dept80
ADD (job_id VARCHAR2(9));
Table altered.
• The new column becomes the last column.
Guidelines for Adding a Column

• You can add or modify columns.
• You cannot specify where the column is to appear. The new column becomes the last column.

Note: If a table already contains rows when a column is added, then the new column is initially null for all the rows.
Modifying a Column

You can modify a column definition by using the ALTER TABLE statement with the MODIFY clause. Column modification can include changes to a column's data type, size, and default value.

Guidelines
• You can increase the width or precision of a numeric column.
• You can increase the width of numeric or character columns.
• You can decrease the width of a column only if the column contains only null values or if the table has no rows.
• You can change the data type only if the column contains null values.
• You can convert a CHAR column to the VARCHAR2 data type or convert a VARCHAR2 column to the CHAR data type only if the column contains null values or if you do not change the size.
• A change to the default value of a column affects only subsequent insertions to the table.

Dropping a Column
You can drop a column from a table by using the ALTER TABLE statement with the DROP COLUMN clause. This is a feature available in Oracle8i and later release.

Guidelines
• The column may or may not contain data.
• Using the ALTER TABLE statement, only one column can be dropped at a time.
• The table must have at least one column remaining in it after it is altered.
• Once a column is dropped, it cannot be recovered.

The SET UNUSED Option
The SET UNUSED option marks one or more columns as unused so that they can be dropped when the demand on system resources is lower. This is a feature available in Oracle8i and later release. Specifying this clause does not actually remove the target columns from each row in the table (that is, it does not
restore the disk space used by these columns). Therefore, the response time is faster than if you executed the DROP clause. Unused columns are treated as if they were dropped, even though their column data remains in the table's rows. After a column has been marked as unused, you have no access to that column.
A SELECT * query will not retrieve data from unused columns. In addition, the names and types of columns marked unused will not be displayed during a DESCRIBE, and you can add to the table a new column with the same name as an unused column. SET UNUSED information is stored in the USER_UNUSED_COL_TABS dictionary view.

The DROP UNUSED COLUMNS Option
DROP UNUSED COLUMNS removes from the table all columns currently marked as unused. You can use
this statement when you want to reclaim the extra disk space fro m unused columns in the table. If the table
contains no unused columns, the statement returns with no errors.

Dropping a Table
The DROP TABLE statement removes the definition of an Oracle table. When you drop a table, the database loses all the data in the table and all the indexes associated with it.
Syntax
DROP TABLE table
In the syntax:
table is the name of the table

Guidelines

• All data is deleted from the table.
• Any views and synonyms remain but are invalid.
• Any pending transactions are committed.
• Only the creator of the table or a user with the DROP ANY TABLE privilege can remove a table.

Note: The DROP TABLE statement, once executed, is irreversible. The Oracle Server does not question the action when you issue the DROP TABLE statement. If you own that table or have a high-level privilege, then the table is immediately removed. As with all DDL statements, DROP TABLE is committed automatically.

Renaming a Table

Additional DDL statements include the RENAME statement, which is used to rename a table, view, sequence, or a synonym.

Syntax

RENAME old_name TO new_name;

In the syntax:

old_name is the old name of the table, view, sequence, or synonym.
new_name is the new name of the table, view, sequence, or synonym.

You must be the owner of the object that you rename.

Truncating a Table

Another DDL statement is the TRUNCATE TABLE statement, which is used to remove all rows from a table and to release the storage space used by that table. When using the TRUNCATE TABLE statement, you cannot rollback row removal.

Syntax

TRUNCATE TABLE table;

In the syntax:

table is the name of the table

You must be the owner of the table or have DELETE TABLE system privileges to truncate a table.

The DELETE statement can also remove all rows from a table, but it does not release storage space. The TRUNCATE command is faster. Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

• The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information.
• Truncating a table does not fire the delete triggers of the table.
• If the table is the parent of a referential integrity constraint, you cannot truncate the table. Disable the constraint before issuing the TRUNCATE statement.

Adding a Comment to a Table

You can add a comment of up to 2,000 bytes about a column, table, view, or snapshot by using the COMMENT statement. The comment is stored in the data dictionary and can be viewed in one of the following data dictionary views in the COMMENTS column:

• ALL_COL_COMMENTS
• USER_COL_COMMENTS
• ALL_TAB_COMMENTS
• USER_TAB_COMMENTS

Syntax
COMMENT ON TABLE table | COLUMN table.column IS 'text';
In the syntax:
table is the name of the table
column is the name of the column in a table
text is the text of the comment
You can drop a comment from the database by setting it to empty string (''): COMMENT ON TABLE employees IS ' ';

<u>Including Constraints</u>

Objectives

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

• Describe constraints
• Create and maintain constraints

Constraints
The Oracle Server uses constraints to prevent invalid data entry into tables. You can use constraints to do the following:
• Enforce rules on the data in a table 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 Oracle Developer

Data Integrity Constraints

Constraint	**Description**
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	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

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 objec t-naming rules. If you do not name your constraint, the Oracle server generates a name with the format SYS_Cn, where n is an integer so that the constraint name is unique. 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.

```
CREATE TABLE employees(employee_id NUMBER(6),first_name VARCHAR2(20),
...
job_id      VARCHAR2(10) NOT NULL,
CONSTRAINT emp_emp_id_pk PRIMARY KEY (EMPLOYEE_ID));
```

In the syntax: *schema is the same as the owner's name*
table is the name of the table
DEFAULT expr specifies a default value to use if a value is omitted in the INSERT statement
column is the name of the column
datatype is the column's data type and length

column_constraint is an integrity constraint as part of the column definition
table_constraint is an integrity constraint as part of the table definition
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	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

The NOT NULL Constraint
The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default.The NOT NULL constraint can be specified only at the column level, not at the table level.

The UNIQUE 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 can have duplicate values in a s pecified 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 constraint comprises more than one column, that group of columns is called a *composite unique key.* UNIQUE 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 colum ns 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 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.

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

Note: The Oracle Server enforces the UNIQUE constraint by implicitly creating a unique index on the unique key column or columns.
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. 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. A table can have only one PRIMARY KEY constraint but can have several UNIQUE constraints.

Note: A UNIQUE index is created automatically for a PRIMARY KEY column.
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, DEPARTMENT_ID has been defined as the foreign key in the EMPLOYEES table (dependent or child table); it references the DEPARTMENT_ID column of the DEPARTMENTS table (the 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.
FOREIGN KEY constraints can be defined at the column or table constraint level. A composite foreign key must be created by using the table-level definition.
The foreign key is defined in the child table, and the table containing the ref erenced 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 co nstraint 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.
• ON DELETE SET NULL converts foreign key values to null when the parent value is removed.
The default behavior is called the restrict rule, which disallow s the update or deletion of referenced data.
Without the ON DELETE CASCADE or the ON DELETE SET NULL options, the row in the parent table cannot be deleted if it is referenced in the child table.

The CHECK Constraint
The CHECK constraint defines a condition that each row must satisfy. The condition c an 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 which reference the column in its definition. There is no limit to the number of CHECK constraints which you can define on a column. CHECK constraints can be defined at the column level or table level.

```
CREATE TABLE employees
(...
salary NUMBER(8,2) CONSTRAINT emp_salary_min
CHECK (salary > 0),
```

Adding a Constraint
You can add a constraint for existing tables by using the ALTER TABLE statement with the ADD clause.
In the syntax:
table is the name of the table
constraint is the name of the constraint
type is the constraint type
column 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.

Note: You can define a NOT NULL column only if the table is empty or if the column has a value for every row.

Disabling a Constraint

You can disable a constraint without dropping it or re-creating it by using the ALTER TABLE statement with the DISABLE clause .

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.
• Disabling a unique or primary key constraint removes the unique index.

Enabling a Constraint

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

Cascading Constraints

This statement illustrates the use of the CASCADE CONSTRAINTS clause. Assume table TEST1 is created as follows:
CREATE TABLE test1 (pk NUMBER PRIMARY KEY, fk NUMBER, col1 NUMBER, col2 NUMBER, CONSTRAINT fk_constraint FOREIGN KEY (fk) REFERENCES test1, CONSTRAINT ck1 CHECK (pk > 0 and col1 > 0), CONSTRAINT ck2 CHECK (col2 > 0));
If all columns referenced by the constraints defined on the dropped columns are also dropped, then CASCADE CONSTRAINTS is not required. For example, assuming that no other referential constraints from other tables refer to column PK, it is valid to submit the following statement without the CASCADE CONSTRAINTS clause:
ALTER TABLE test1 DROP (pk, fk, col1);

Viewing Constraints

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.

Viewing the Columns Associated with Constraints

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