Oracle Database 11*g*: SQL Fundamentals I

Volume I • Student Guide

D49996GC10

Edition 1.0

August 2007

D52128

ORACLE®

Author

Puja Singh

Technical Contributors and Reviewers

Claire Bennett
Tom Best
Purjanti Chang
Ken Cooper
László Czinkóczki
Burt Demchick
Mark Fleming
Gerlinde Frenzen
Nancy Greenberg
Chaitanya Koratamaddi
Wendy Lo
Timothy Mcglue
Alan Paulson
Bryan Roberts
Abhishek Singh
Lori Tritz
Michael Versaci
Lex van der Werff

Graphic Designers

Satish Bettegowda
Samir Mozumdar

Editors

Amitha Narayan
Vijayalakshmi Narasimhan

Publisher

Sujatha Nagendra

Index

Preface

Profile

Before You Begin This Course

Before you begin this course, you should be able to use a graphical user interface (GUI). The prerequisite is a familiarity with data processing concepts and techniques.

How This Course Is Organized

Oracle Database 11g: SQL Fundamentals I is an instructor-led course featuring lectures and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills that are introduced.

Related Publications

Oracle Publications

Title	Part Number
Oracle® Database Reference 11g Release 1 (11.1)	B28320-01
Oracle® Database SQL Language Reference 11g Release 1 (11.1)	B28286-01
Oracle® Database Concepts 11g Release 1 (11.1)	B28318-01
Oracle® Database SQL Developer User's Guide Release 1.2	E10406-01

Additional Publications

- System release bulletins
- Installation and user's guides
- *read.me* files
- International Oracle User's Group (IOUG) articles
- *Oracle Magazine*

Typographic Conventions

What follows are two lists of typographical conventions that are used specifically within text or within code.

Typographic Conventions Within Text

Convention	Object or Term	Example
Uppercase	Commands, functions, column names, table names, PL/SQL objects, schemas	Use the `SELECT` command to view information stored in the `LAST_NAME` column of the `EMPLOYEES` table.
Lowercase, italic	Filenames, syntax variables, usernames, passwords	**where:** *role* is the name of the role to be created.
Initial cap	Trigger and button names	Assign a When-Validate-Item trigger to the ORD block. Choose Cancel.
Italic	Books, names of courses and manuals, and emphasized words or phrases	For more information on the subject see *Oracle SQL Reference Manual* Do *not* save changes to the database.
Quotation marks	Lesson module titles referenced within a course	This subject is covered in Lesson 3, “Working with Objects.”

Typographic Conventions (continued)

Typographic Conventions Within Code

Convention	Object or Term	Example
Uppercase	Commands, functions	`SELECT employee_id` `FROM  employees;`
Lowercase, italic	Syntax variables	`CREATE ROLE` ***`role`***`;`

Initial cap	Forms triggers	`Form module: ORD`
		`Trigger level: S_ITEM.QUANTITY item`
		`Trigger name: When-Validate-Item`
		`. . .`

Lowercase	Column names, table names, filenames, PL/SQL objects	`. . .`
		`OG_ACTIVATE_LAYER`
		` (OG_GET_LAYER ('prod_pie_layer'))`
		`. . .`
		
		`SELECT last_name`
		`FROM   employees;`

Bold	Text that must be entered by a user	`CREATE USER scott` `IDENTIFIED BY tiger;`

I

Introduction

Lesson Objectives

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

- Understand the goals of the course
- List the features of Oracle Database 11*g*
- Discuss the theoretical and physical aspects of a relational database
- Describe Oracle server’s implementation of RDBMS and object relational database management system (ORDBMS)
- Identify the development environments that can be used for this course
- Describe and use the key features of Oracle SQL Developer
- Describe the database and schema used in this course

ORACLE

I - 2

Objectives

In this lesson, you gain an understanding of the relational database management system (RDBMS) and the object relational database management system (ORDBMS). You are also introduced to Oracle SQL Developer and SQL*Plus as development environments used for executing SQL statements, and for formatting and reporting purposes.

Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 11*g* and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- Overview of Oracle SQL Developer
- Oracle Database 11*g* documentation and additional resources

ORACLE

Course Objectives

After completing this course, you should be able to:

- Identify the major components of Oracle Database 11*g*
- Retrieve row and column data from tables with the `SELECT` statement
- Create reports of sorted and restricted data
- Employ SQL functions to generate and retrieve customized data
- Run complex queries to retrieve data from multiple tables
- Run data manipulation language (DML) statements to update data in Oracle Database 11*g*
- Run data definition language (DDL) statements to create and manage schema objects

ORACLE

I - 4

Course Objectives

This course offers you an introduction to Oracle Database 11*g* database technology. In this class, you learn the basic concepts of relational databases and the powerful SQL programming language. This course provides the essential SQL skills that enable you to write queries against single and multiple tables, manipulate data in tables, create database objects, and query metadata.

Course Agenda

- Day 1:
  - Introduction
  - Retrieving Data Using the SQL `SELECT` Statement
  - Restricting and Sorting Data
  - Using Single-Row Functions to Customize Output
  - Using Conversion Functions and Conditional Expressions
- Day 2:
  - Reporting Aggregated Data Using the Group Functions
  - Displaying Data from Multiple Tables
  - Using Subqueries to Solve Queries
  - Using the Set Operators

Course Agenda

- Day 3:
  - Manipulating Data
  - Using DDL Statements to Create and Manage Tables
  - Creating Other Schema Objects

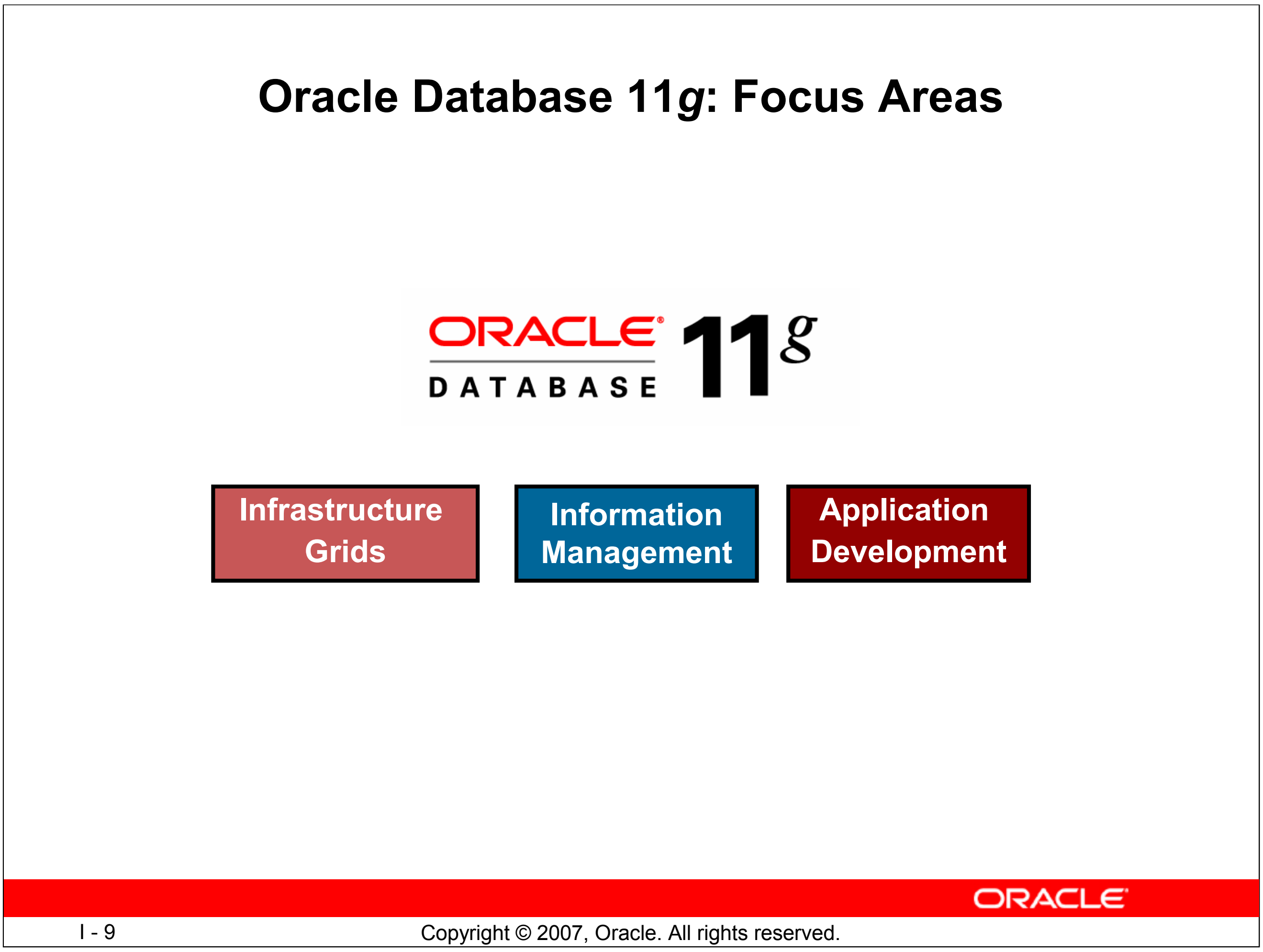
Appendixes Used in the Course

- Appendix A: Practice Solutions
- Appendix B: Table Descriptions
- Appendix C: Oracle Join Syntax
- Appendix D: Using SQL*Plus
- Appendix E: Performing DML and DDL Operations Using the Oracle SQL Developer GUI
- Additional Practices
- Additional Practices Solutions

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11*g* and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- Overview of Oracle SQL Developer
- The HR schema and the tables used in this course
- Oracle Database 11*g* documentation and additional resources

ORACLE



Oracle Database 11*g*: Focus Areas

Oracle Database 11*g* offers extensive features across the following focus areas:

- **Infrastructure Grids:** The Infrastructure Grid technology of Oracle enables pooling of low-cost servers and storage to form systems that deliver the highest quality of service in terms of manageability, high availability, and performance. Oracle Database 11*g* consolidates and extends the benefits of grid computing. Apart from taking full advantage of grid computing, Oracle Database 11*g* has unique change assurance features to manage changes in a controlled and cost effective manner.
- **Information Management:** Oracle Database 11*g* extends the existing information management capabilities in content management, information integration, and information life cycle management areas. Oracle provides content management of advanced data types such as Extensible Markup Language (XML), text, spatial, multimedia, medical imaging, and semantic technologies.
- **Application Development:** Oracle Database 11*g* has capabilities to use and manage all the major application development environments such as PL/SQL, Java/JDBC, .NET and Windows, PHP, SQL Developer, and Application Express.

Oracle Database 11*g*

Manageability

High availability

Performance

Security

Information integration

Oracle Database 11*g*

Organizations need to support multiple terabytes of information for users who demand fast and secure access to business applications round-the-clock. The database systems must be reliable and must be able to recover quickly in the event of any kind of failure. Oracle Database 11*g* is designed along the following feature areas to help organizations manage infrastructure grids easily and deliver high-quality service:

- **Manageability:** By using some of the change assurance, management automation, and fault diagnostics features, the database administrators (DBAs) can increase their productivity, reduce costs, minimize errors, and maximize quality of service. Some of the useful features that promote better management are Database Replay facility, the SQL Performance Analyzer, and the Automatic SQL Tuning facility.
- **High availability:** By using the high availability features, you can reduce the risk of down time and data loss. These features improves online operations and enable faster database upgrades.

Oracle Database 11*g*

Manageability

High availability

Performance

Security

Information integration

Oracle Database 11*g* (continued)

- **Performance:** By using capabilities such as SecureFiles, compression for online transaction processing (OLTP), Real Application Clusters (RAC) optimizations, Result Caches and so on, you can greatly improve the performance of your database. Oracle Database 11*g* enables organizations to manage large, scalable transactional and data warehousing systems that deliver fast data access using low-cost modular storage.
- **Security:** Oracle Database 11*g* helps organizations protect their information with unique secure configurations, data encryption and masking, and sophisticated auditing capabilities. It delivers a secure and scalable platform for reliable and fast access to all types of information by using the industry-standard interfaces.
- **Information integration:** Oracle Database 11*g* has many features to better integrate data throughout the enterprise. It also supports advanced information life cycle management capabilities. This helps you manage the changing data in your database.

Oracle Fusion Middleware

Portfolio of leading, standards-based, and customer-proven software products that spans a range of tools and services from J2EE and developer tools, through integration services, business intelligence, collaboration, and content management

I - 12

Oracle Fusion Middleware

Oracle Fusion Middleware is a comprehensive and well-integrated family of products that offers complete support for development, deployment, and management of Service-Oriented Architecture (SOA). SOA facilitates the development of modular business services that can be easily integrated and reused, thereby reducing development and maintenance costs, and providing higher quality of services. Oracle Fusion Middleware's pluggable architecture enables you to leverage your investments in any existing application, system, or technology. Its unbreakable core technology minimizes the disruption caused by planned or unplanned outages.

Some of the products from the Oracle Fusion Middleware family include:

- **Enterprise Application Server:** Application Server
- **Integration and Process Management:** BPEL Process Manager, Oracle Business Process Analysis Suite
- **Development Tools:** Oracle Application Development Framework, JDeveloper, SOA Suite
- **Business Intelligence:** Oracle Business Activity Monitoring, Oracle Data Integrator
- **Systems Management:** Enterprise Manager
- **Identity Management:** Oracle Identity Management
- **Content Management:** Oracle Content Database Suite
- **User Interaction:** Portal, WebCenter

Oracle Enterprise Manager Grid Control 10*g*

- Efficient Oracle Fusion Middleware management
- Simplifying application and infrastructure life cycle management
- Improved database administration and application management capabilities

ORACLE
ENTERPRISE MANAGER 10*g*

ORACLE

I - 13

Oracle Enterprise Manager Grid Control 10*g*

Spanning applications, middleware, and database management, Oracle Enterprise Manager Grid Control 10*g* delivers integrated enterprise management for Oracle and non-Oracle systems.

Oracle Enterprise Manager Grid Control 10*g* features advanced Oracle Fusion Middleware management capabilities for the services that business applications rely upon, including SOA, Business Activity Monitoring, and Identity Management.

- **Wide-ranging management functionality** for your applications including service-level management, application performance management, configuration management, and change automation
- **Built-in grid automation capabilities** means that information technology responds proactively to fluctuating demand and implements new services more quickly so that businesses can thrive.
- **In-depth diagnostics and readily available remediation** across a range of applications including custom-built applications, Oracle E-Business Suite, PeopleSoft, Siebel, Oracle Fusion Middleware, Oracle Database, and underlying infrastructure
- **Extensive life cycle management capabilities** extends grid computing by providing solutions for the entire application and infrastructure life cycle, including test, stage, and production through operations. It has simplified patch management with synchronized patching, additional operating system support, and conflict detection features.

Oracle BI Publisher

- Provides a central architecture for authoring, managing, and delivering information in secure and multiple formats
- Reduces complexity and time to develop, test, and deploy all kinds of reports
  – Financial Reports, Invoices, Sales or Purchase orders, XML, and EDI/EFT(eText documents)
- Enables flexible customizations
  – For example, a Microsoft Word document report can be generated in multiple formats such as PDF, HTML, Excel, RTF, and so on.

I - 14

Oracle BI Publisher

Oracle Database 11*g* also includes Oracle BI Publisher—the enterprise reporting solution from Oracle. Oracle BI Publisher (formerly known as XML Publisher) offers the most efficient and scalable reporting solution available for complex, distributed environments.

Oracle BI Publisher reduces the high costs associated with the development, customization, and maintenance of business documents, while increasing the efficiency of reports management. By using a set of familiar desktop tools, users can create and maintain their own report formats based on data queries created by the IT staff or developers.

Oracle BI Publisher report formats can be designed using Microsoft Word or Adobe Acrobat—tools that most users are already familiar with. Oracle BI Publisher also enables you to bring in data from multiple data sources into a single output document. You can deliver reports via printer, email, or fax. You can publish your report to a portal. You can even allow users to collaboratively edit and manage reports on the Web-based Distributed Authoring and Versioning (WebDav) Web servers.

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11*g* and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- Overview of Oracle SQL Developer
- The HR schema and the tables used in this course
- Oracle Database 11*g* documentation and additional resources

ORACLE

Relational and Object Relational Database Management Systems

- Relational model and object relational model
- User-defined data types and objects
- Fully compatible with relational database
- Supports multimedia and large objects
- High-quality database server features

I - 16

Relational and Object Relational Database Management Systems

The Oracle server supports both the relational and the object relational database models.

The Oracle server extends the data-modeling capabilities to support an object relational database model that provides object-oriented programming, complex data types, complex business objects, and full compatibility with the relational world.

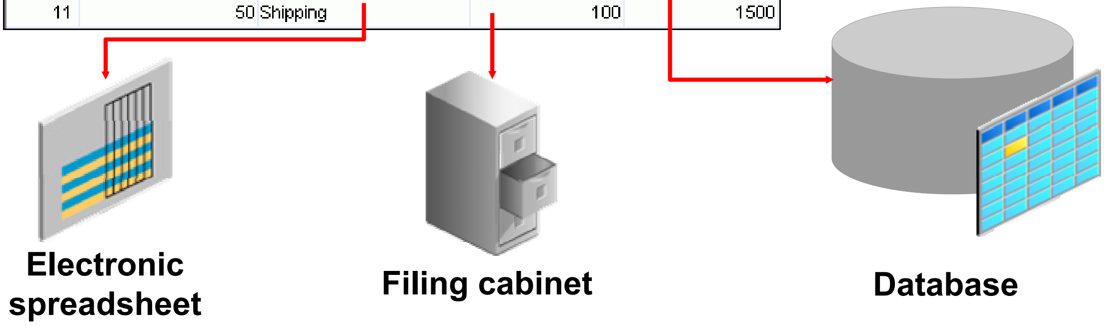
It includes several features for improved performance and functionality of the OLTP applications, such as better sharing of run-time data structures, larger buffer caches, and deferrable constraints. Data warehouse applications benefit from enhancements such as parallel execution of insert, update, and delete operations; partitioning; and parallel-aware query optimization. Operating within the Network Computing Architecture (NCA) framework, the Oracle model supports client/server and Web-based applications that are distributed and multitiered.

For more information about the relational and object relational model, see the *Oracle Database Concepts 11g Release 1 (11.1)* manual.

Data Storage on Different Media

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	101	1700
2	20	Marketing		
3	20	Marketing		
4	30	Purchasing		
5	30	Purchasing		
6	30	Purchasing		
7	30	Purchasing		
8	30	Purchasing		
9	30	Purchasing	114	1700
10	40	Human Resources	101	2400
11	50	Shipping	100	1500

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000



Data Storage on Different Media

Every organization has some information needs. A library keeps a list of members, books, due dates, and fines. A company needs to save information about its employees, departments, and salaries. These pieces of information are called *data*.

Organizations can store data in various media and in different formats, such as a hard copy document in a filing cabinet, or data stored in electronic spreadsheets, or in databases.

A *database* is an organized collection of information.

To manage databases, you need a database management system (DBMS). A DBMS is a program that stores, retrieves, and modifies data in databases on request. There are four main types of databases: *hierarchical*, *network*, *relational*, and (most recently) *object relational*.

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 (RDBMS).
- 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

ORACLE

Relational Database Concept

The principles of the relational model were first outlined by Dr. E. F. Codd in a June 1970 paper titled "A Relational Model of Data for Large Shared Data Banks." In this paper, Dr. Codd proposed the relational model for database systems.

The common models used at that time were hierarchical and network, or even simple flat-file data structures. Relational database management systems (RDBMS) soon became very popular, especially for their ease of use and flexibility in structure. In addition, a number of innovative vendors, such as Oracle, supplemented the RDBMS with a suite of powerful, application development and user-interface products, thereby providing a total solution.

Components of the Relational Model

- Collections of objects or relations that store the data
- A set of operators that can act on the relations to produce other relations
- Data integrity for accuracy and consistency

For more information, see *An Introduction to Database Systems, Eighth Edition* (Addison-Wesley: 2004), written by Chris Date.

Definition of a Relational Database

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

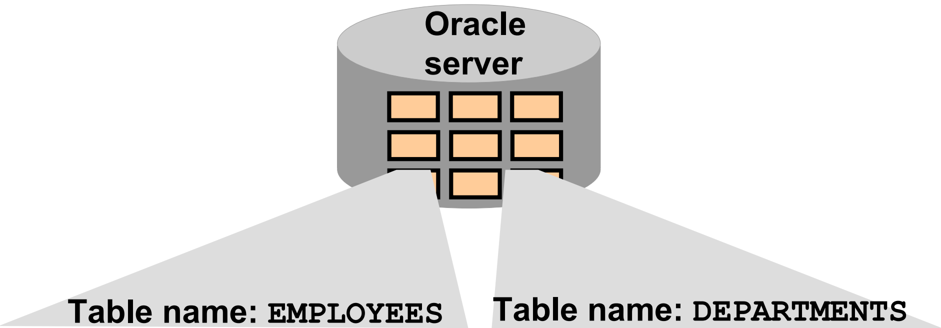


Table name: `EMPLOYEES`

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

…

Table name: `DEPARTMENTS`

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID
10	Administration	200
20	Marketing	201
50	Shipping	124

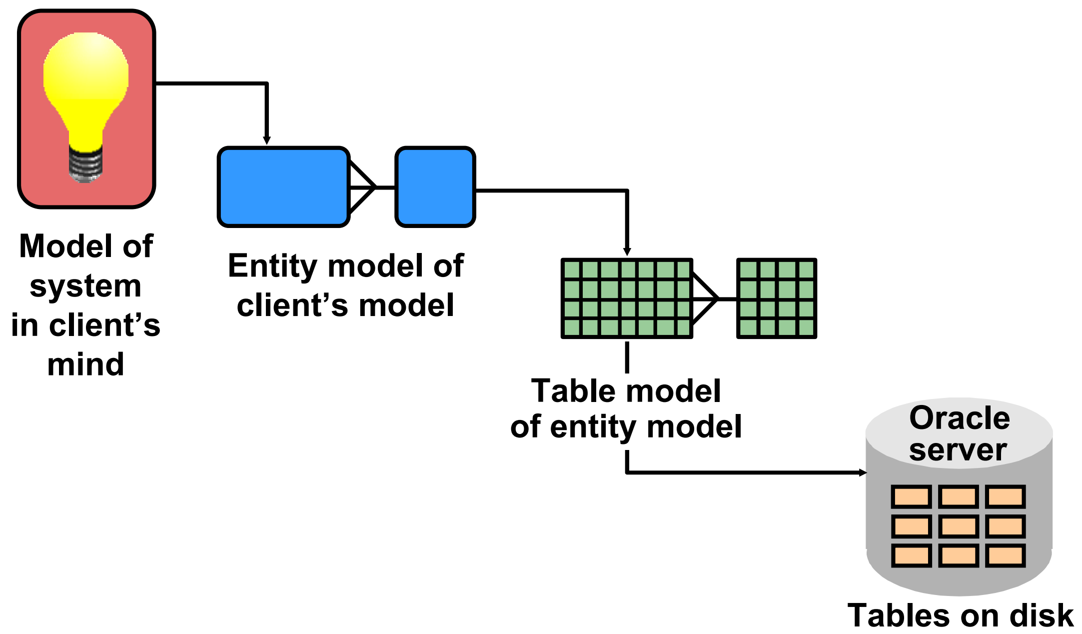
…

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.

Data Models



Data Models

Models are the cornerstone of design. Engineers build a model of a car to work out any details before putting it into production. In the same manner, system designers develop models to explore ideas and improve the understanding of database design.

Purpose of Models

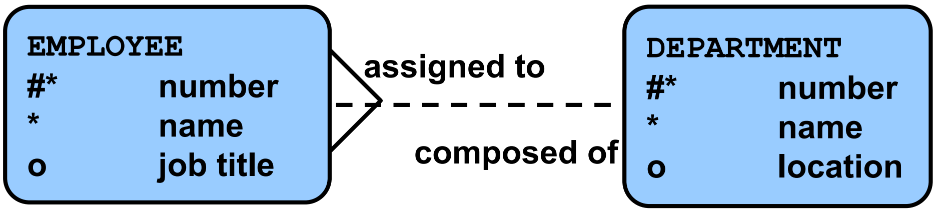
Models help communicate the concepts that are in people's minds. They can be used to do the following:

- Communicate
- Categorize
- Describe
- Specify
- Investigate
- Evolve
- Analyze
- Imitate

The objective is to produce a model that fits a multitude of these uses, can be understood by an end user, and contains sufficient detail for a developer to build a database system.

Entity Relationship Model

- Create an entity relationship diagram from business specifications or narratives:



- Scenario:
  – "... Assign one or more employees to a department ..."
  – "... Some departments do not yet have assigned employees ..."

I - 21

Entity Relationship Model

In an effective system, data is divided into discrete categories or entities. An entity relationship (ER) model is an illustration of the various entities in a business and the relationships among 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 the business. Although businesses can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant.

Entity Relationship Model (continued)

Benefits of ER Modeling:

- Documents information for the organization in a clear, precise format
- Provides a clear picture of the scope of the information requirement
- Provides an easily understood pictorial map for database design
- Offers an effective framework for integrating multiple applications

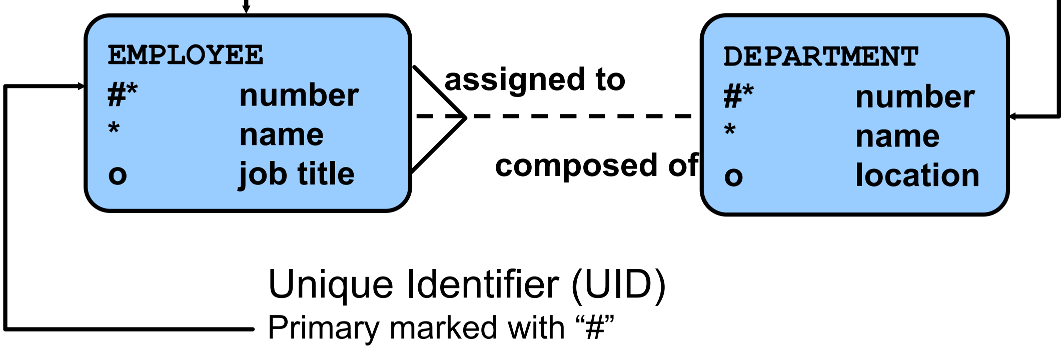
Key Components

- **Entity:** An aspect of significance about which information must be known. 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

Entity Relationship Modeling Conventions

Entity:
- Singular, unique name
- Uppercase
- Soft box
- Synonym in parentheses

Attribute:
- Singular name
- Lowercase
- Mandatory marked with "*"
- Optional marked with "o"



Unique Identifier (UID)
Primary marked with "#"
Secondary marked with "(#)"

ORACLE

ER Modeling Conventions

Entities

To represent an entity in a model, use the following conventions:
- Singular, unique entity name
- Entity name in uppercase
- Soft box
- Optional synonym names in uppercase within parentheses: ( )

Attributes

To represent an attribute in a model, use the following conventions:
- Singular name in lowercase
- Asterisk (*) tag for mandatory attributes (that is, values that *must* be known)
- Letter "o" tag for optional attributes (that is, values that *may* be known)

Relationships

Symbol	Description
Dashed line	Optional element indicating "maybe"
Solid line	Mandatory element indicating "must be"
Crow's foot	Degree element indicating "one or more"
Single line	Degree element indicating "one and only one"

ER Modeling Conventions (continued)

Relationships

Each direction of the relationship contains:

- **A label:** for example, *taught by* or *assigned to*
- **An optionality:** either *must be* or *maybe*
- **A degree:** either *one and only one* or *one or more*

Note: The term *cardinality* is a synonym for the term *degree*.

Each source entity {may be | must be} in relation {one and only one | one or more} with the destination entity.

Note: The convention is to read clockwise.

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 hash sign (#).
- Tag secondary UIDs with a hash sign in parentheses (#).

Relating Multiple Tables

- Each row of data in a table is uniquely identified by a primary key.
- You can logically relate data from multiple tables using foreign keys.

Table name: `DEPARTMENTS`

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting	(null)	1700

Table name: `EMPLOYEES`

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
103	Alexander	Hunold	60
104	Bruce	Ernst	60
107	Diana	Lorentz	60

…

Primary key **Foreign key** **Primary key**

ORACLE

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. By 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 refers 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 the department data by storing it in a separate table.

Relating Multiple Tables (continued)

Guidelines for Primary Keys and Foreign Keys

- You cannot 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, otherwise it must be null.
- A foreign key must reference either a primary key or a unique key column.

Relational Database Terminology

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	90
101	Neena	Kochhar	17000	(null)	90
102	Lex	De Haan	17000	(null)	90
103	Alexander	Hunold	9000	(null)	60
104	Bruce	Ernst	6000	(null)	60
107	Diana	Lorentz	4200	(null)	60
124	Kevin	Mourgos	5800	(null)	50
141	Trenna	Rajs	3500	[illegible]	50
142	Curtis	Davies	3100	(null)	50
143	Randall	Matos	2600	(null)	50
144	Peter	Vargas	2500	(null)	50
149	Eleni	Zlotkey	10500	0.2	80
174	Ellen	Abel	11000	0.3	80
176	Jonathon	Taylor	8600	0.2	80
178	Kimberely	Grant	7000	0.15	(null)
200	Jennifer	Whalen	4400	(null)	10
201	Michael	Hartstein	13000	(null)	20
202	Pat	Fay	6000	(null)	20
205	Shelley	Higgins	12000	(null)	110
206	William	Gietz	8300	(null)	110

ORACLE

Relational Database Terminology

A relational database can contain one or many tables. A *table* is the basic storage structure of an RDBMS. A table holds all the data necessary about something in the real world, such as employees, invoices, or customers.

The slide shows the contents of the `EMPLOYEES` *table* or *relation*. The numbers indicate the following:

1. A single *row* (or *tuple*) representing all the data required for a particular employee. Each row in a table should be identified by a primary key, which permits no duplicate rows. The order of rows is insignificant; specify the row order when the data is retrieved.
2. A *column* or attribute containing the employee number. The employee number identifies a *unique* employee in the `EMPLOYEES` table. In this example, the employee number column is designated as the *primary key*. A primary key must contain a value and the value must be unique.
3. A column that is not a key value. A column represents one kind of data in a table; in this example, the data is the salaries of all the employees. Column order is insignificant when storing data; specify the column order when the data is retrieved.

Relational Database Terminology (continued)

4. A column containing the department number, which is also a *foreign key*. A foreign key is a column that defines how tables relate to each other. A foreign key refers to a primary key or a unique key in the same table or in another table. In the example, `DEPARTMENT_ID` uniquely identifies a department in the `DEPARTMENTS` table.
5. A *field* can be found at the intersection of a row and a column. There can be only one value in it.
6. A field may have no value in it. This is called a null value. In the `EMPLOYEES` table, only those employees who have the role of sales representative have a value in the `COMMISSION_PCT` (commission) field.

Lesson Agenda

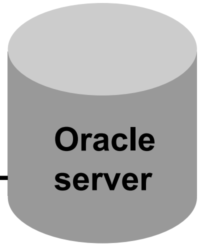
- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11*g* and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- Overview of Oracle SQL Developer
- The HR schema and the tables used in this course
- Oracle Database 11*g* documentation and additional resources

Using SQL to Query Your Database

Structured query language (SQL) is:

- The ANSI standard language for operating relational databases
- Efficient, easy to learn, and use
- Functionally complete (With SQL, you can define, retrieve, and manipulate data in the tables.)

```
SELECT department_name
FROM   departments;
```



DEPARTMENT_NAME
Administration
Marketing
Shipping
IT
Sales
Executive
Accounting
Contracting

ORACLE

Using SQL to Query Your 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. SQL is a set of statements with which all programs and users access data in an Oracle database. Application programs and Oracle tools often allow users access to the database without using SQL directly, but these applications, in turn, must use SQL when executing the user's request.

SQL provides statements for a variety of tasks, including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects
- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

SQL unifies all of the preceding tasks in one consistent language and enables you to work with data at a logical level.

SQL Statements

SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control

SQL Statements

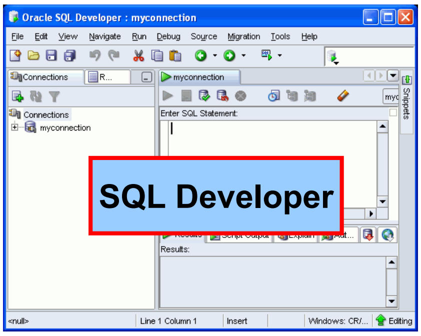
SQL statements supported by Oracle comply with industry standards. Oracle Corporation ensures future compliance with evolving standards by actively involving key personnel in SQL standards committees. The industry-accepted committees are ANSI and International Standards Organization (ISO). Both ANSI and ISO have accepted SQL as the standard language for relational databases.

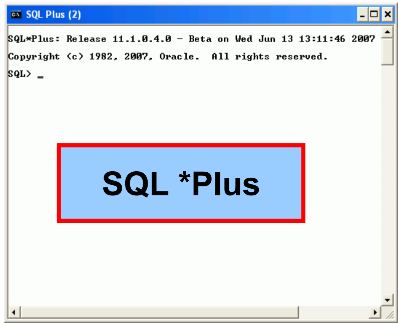
Statement	Description
SELECT INSERT UPDATE DELETE MERGE	Retrieves data from the database, enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively. Collectively known as *data manipulation language* (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Sets up, changes, and removes data structures from tables. Collectively known as *data definition language* (DDL)
GRANT REVOKE	Provides or removes access rights to both the Oracle database and the structures within it
COMMIT ROLLBACK SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions

Development Environments for SQL

In this course:

- Primarily use Oracle SQL Developer, Release 1.2
- Use SQL*Plus:
  - In case you do not have access to Oracle SQL Developer
  - Or when any command does not work in Oracle SQL Developer





I - 32

Development Environments for SQL

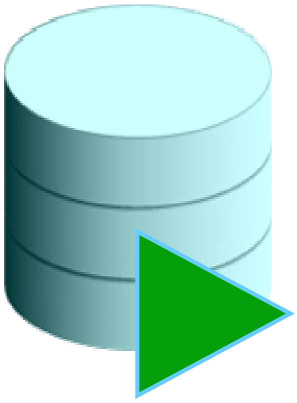
This course has been developed using Oracle SQL Developer as the tool for running the SQL statements discussed in the examples in the slide and the practices. For commands that are not supported by Oracle SQL Developer, use the SQL*Plus environment.

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11*g* and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- Overview of Oracle SQL Developer
- The HR schema and the tables used in this course
- Oracle Database 11*g* documentation and additional resources

What Is Oracle SQL Developer?

- Oracle SQL Developer is a graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema by using the standard Oracle database authentication.



SQL Developer

ORACLE

I - 34

What Is Oracle SQL Developer?

Oracle SQL Developer is a free graphical tool designed to improve your productivity and simplify the development of everyday database tasks. With just a few clicks, you can easily create and debug stored procedures, test SQL statements, and view optimizer plans.

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

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

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

Note: The Oracle SQL Developer, Release 1.2 is called the *Migration release* because it tightly integrates with *Developer Migration Workbench*. Therefore, it provides users with a single point to browse database objects and data in third-party databases, and to migrate from these databases to Oracle. You can also connect to schemas for selected third-party (non-Oracle) databases, such as MySQL, Microsoft SQL Server, and Microsoft Access, and view metadata and data in these databases.

Additionally, Oracle SQL Developer, Release 1.2 includes support for Oracle Application Express, Release 3.0.1 (Oracle APEX).

Specifications of Oracle SQL Developer

- Developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Default connectivity by using the JDBC Thin driver
- Does not require an installer
  - Unzip the downloaded Oracle SQL Developer kit and double-click `sqldeveloper.exe` to start Oracle SQL Developer.
- Connects to any Oracle Database, version 9.2.0.1 and later
- Freely downloadable from the following link:
  - http://www.oracle.com/technology/products/database/sql_developer/index.html
- Needs JDK 1.5 installed on your system, which can be downloaded from the following link:
  - http://java.sun.com/javase/downloads/index_jdk5.jsp

ORACLE

I - 35

Specifications of Oracle SQL Developer

Oracle SQL Developer is developed in Java leveraging the Oracle JDeveloper integrated development environment (IDE). It is a cross-platform tool. The tool runs on Windows, Linux, and Mac operating system (OS) X platforms. You can install Oracle SQL Developer on the database server and connect remotely from your desktop, thus avoiding client/server network traffic.

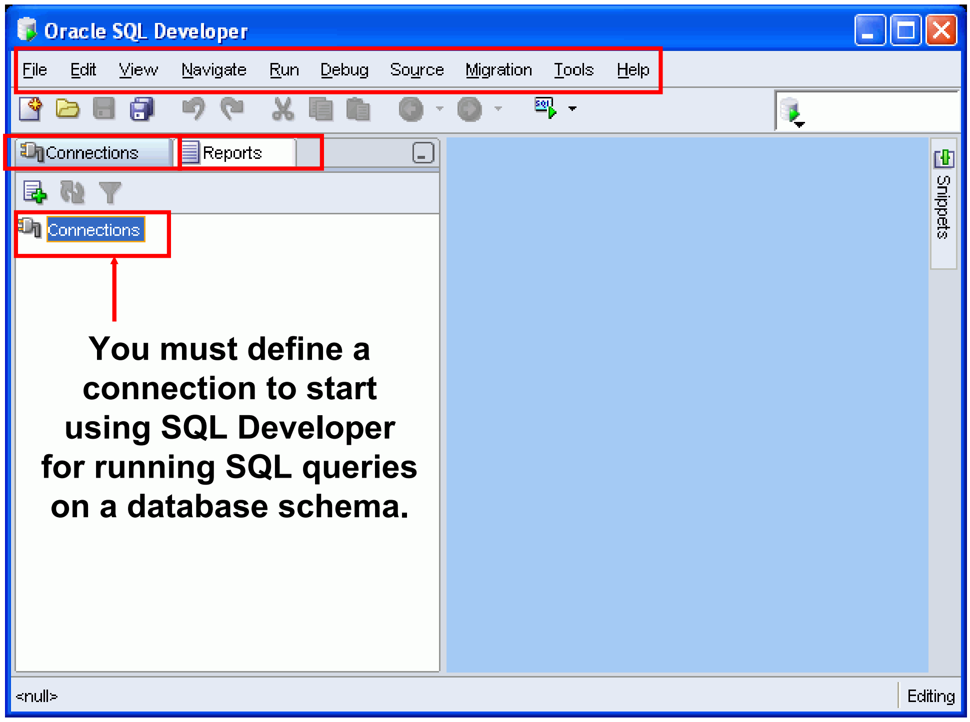
Default connectivity to the database is through the Java Database Connectivity (JDBC) Thin driver, so Oracle Home is not required. Oracle SQL Developer does not require an installer and you need to simply unzip the downloaded file. With Oracle SQL Developer, users can connect to the Oracle Database, version 9.2.0.1 and later, and all Oracle database editions including the Express Edition.

Oracle SQL Developer can be downloaded with the following packaging options:

- Oracle SQL Developer for Windows (option to download with or without Java Development Kit (JDK) 1.5)
- Oracle SQL Developer for Multiple Platforms (you should have JDK 1.5 already installed)
- Oracle SQL Developer for Mac OS X platforms (you should have JDK 1.5 already installed)
- Oracle SQL Developer RPM for Linux (you should have JDK 1.5 already installed)

Note that Oracle SQL Developer, Release 1.2 is also certified with JDK 6.0.

Oracle SQL Developer Interface



ORACLE

Oracle SQL Developer Interface

Oracle SQL Developer has two main navigation tabs:

- **Connections tab:** By using this tab, you can browse database objects and users to which you have access.
- **Reports tab:** By using this tab, you can run predefined reports, or create and add your own reports.

Oracle SQL Developer uses the left pane for navigation to find and select objects, and the right pane to display information about selected objects. You can customize many aspects of the appearance and behavior of Oracle SQL Developer by setting preferences. The menus at the top contain standard entries, plus entries for features specific to Oracle SQL Developer.

1. **View:** Contains options that affect what is displayed in the Oracle SQL Developer interface
2. **Navigate:** Contains options for navigating to panes and in the execution of subprograms
3. **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected
4. **Debug:** Contains options relevant when a function or procedure is selected for debugging
5. **Source:** Contains options for use when editing functions and procedures
6. **Migration:** Contains options related to migrating third-party databases to Oracle
7. **Tools:** Invokes tools such as SQL*Plus, Preferences, and SQL Worksheet

Note: You must define at least one connection to be able to connect to a database schema and issue SQL queries or run procedures/functions.

Creating a Database Connection

- You must have at least one database connection to use Oracle SQL Developer.
- You can create and test connections for:
  - Multiple databases
  - Multiple schemas
- Oracle SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an XML file.
- Each additional database connection created is listed in the Connections Navigator hierarchy.

ORACLE

I - 37

Creating a Database Connection

A connection is an Oracle SQL Developer object that specifies the necessary information for connecting to a specific database as a specific user of that database. To use Oracle SQL Developer, you must have at least one database connection, which may be existing, created, or imported.

You can create and test connections for multiple databases and for multiple schemas.

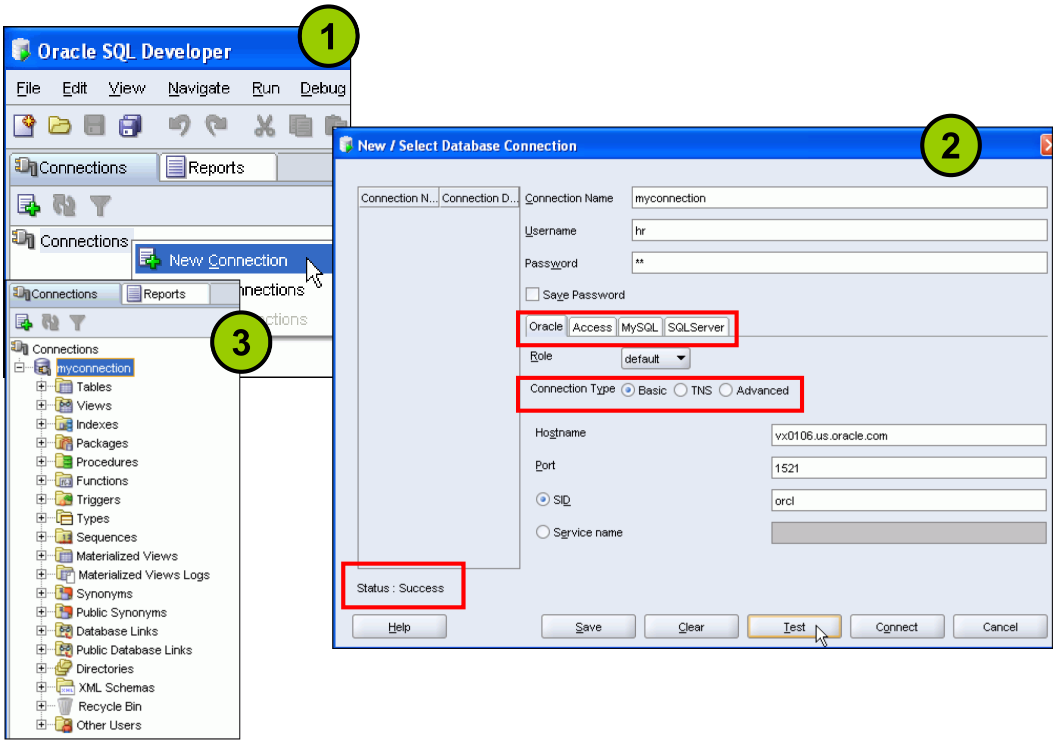
By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory. But, it can also be in the directory specified by the `TNS_ADMIN` environment variable or the registry value. When you start Oracle SQL Developer and display the Database Connections dialog box, Oracle SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.

Note: On Windows systems, if the `tnsnames.ora` file exists but Oracle SQL Developer is not using its connections, define `TNS_ADMIN` as a system environment variable.

You can export connections to an XML file so that you can reuse it later.

You can create additional connections as different users to the same database or to connect to the different databases.

Creating a Database Connection



Creating a Database Connection (continued)

To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click Connections and select New Connection.
2. In the New/Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
    1. From the Role drop-down list, you can select either *default* or SYSDBA (you will select SYSDBA for the `sys` user or any user with DBA privileges).
    2. You can select the connection type as:
        - Basic: In this type, you enter the host name and system identifier (SID) for the database that you want to connect to. The Port is already set to 1521. Or, you can also enter the Service name directly if you are using a remote database connection.
        - TNS: You select any one of the database aliases imported from the `tnsnames.ora` file
        - Advanced: You define a custom JDBC URL to connect to the database.
    3. Click Test to make sure that the connection has been set correctly.
    4. Click Connect.

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

Creating a Database Connection (continued)

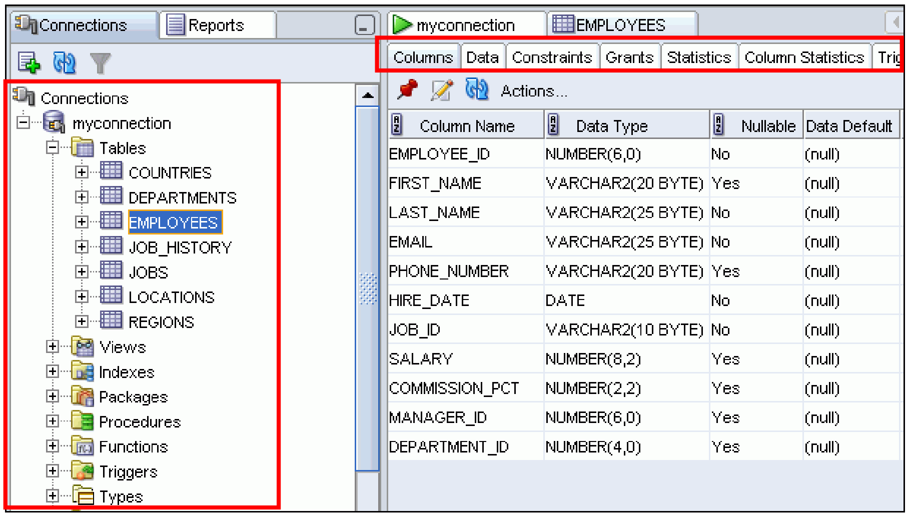
3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions, for example, dependencies, details, statistics, and so on.

Note: From the same New/Select Database Connection window, you can define connections to non-Oracle data sources by using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Browsing Database Objects

Use the Connections Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



Browsing Database Objects

After you have created a database connection, you can use the Connections Navigator to browse through many objects in a database schema including Tables, Views, Indexes, Packages, Procedures, Triggers, Types, and so on.

Oracle SQL Developer uses the left pane for navigation to find and select objects and the right pane to display information about the selected objects. You can customize many aspects of the appearance of Oracle SQL Developer by setting preferences.

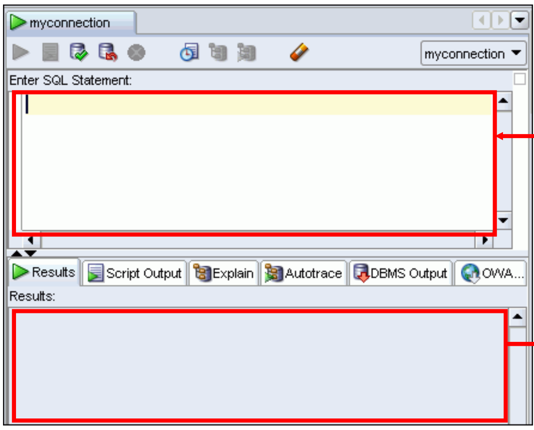
You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, the details about columns, constraints, grants, statistics, triggers, and so on are displayed in an easy-to-read tabbed page.

If you want to see the definition of the `EMPLOYEES` table as shown in the slide, perform the following steps:

1. Expand the Connections node in the Connections Navigator.
2. Expand Tables.
3. Click EMPLOYEES. By default, the Columns tab is selected. It shows the column description of the table. By using the Data tab, you can view the tables data and also enter new rows, update data, and commit these changes to the database.

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements.
- Specify any actions that can be processed by the database connection associated with the Worksheet.



Enter SQL Statements.

Results are shown here.

Using the SQL Worksheet

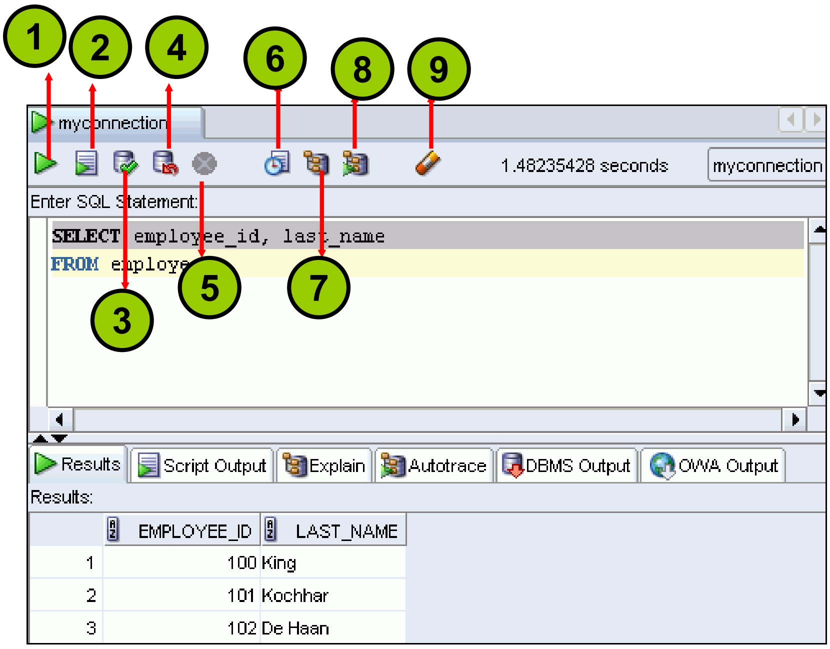
When you connect to a database, a SQL Worksheet window for that connection is automatically opened. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from the SQL Worksheet to the Oracle database. However, the SQL*Plus commands used in Oracle SQL Developer have to be interpreted by the SQL Worksheet before being passed to the database.

The SQL Worksheet currently supports a number of SQL*Plus commands. Those commands that are not supported by the SQL Worksheet are ignored and not sent to the Oracle database. Through the SQL Worksheet, you can execute SQL statements and some of the SQL*Plus commands.

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

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon available on the main toolbar.

Using the SQL Worksheet



Using the SQL Worksheet (continued)

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

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

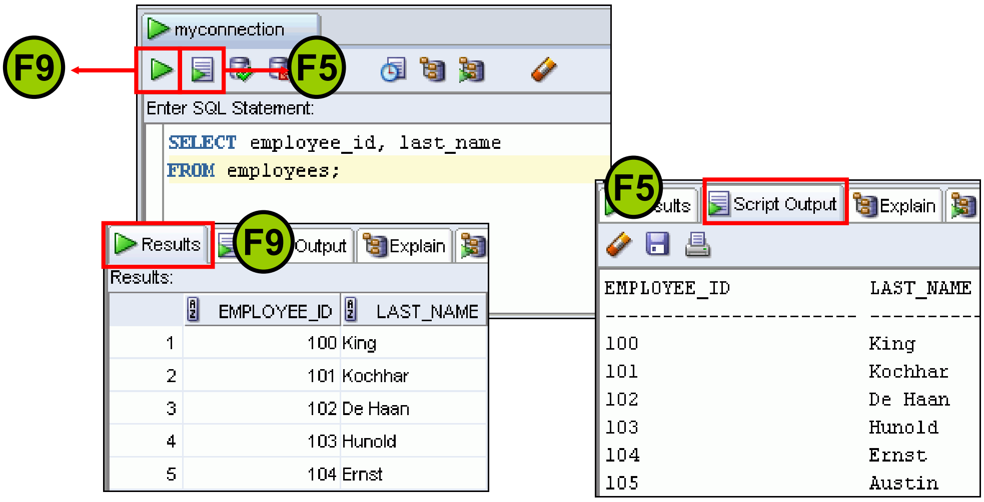
1. **Execute Statement:** This executes the statement at the cursor in the Enter SQL Statement box. Alternatively, you can press [F9]. The output is generally shown in a formatted manner in the Results tab page.
2. **Run Script:** This executes all statements in the Enter SQL Statement box using the Script Runner. The output is generally shown in the conventional script format in the Scripts tab page.
3. **Commit:** This writes any changes to the database and ends the transaction.
4. **Rollback:** This discards any changes to the database, without writing them to the database, and ends the transaction.
5. **Cancel:** This stops the execution of any statements currently being executed.

Using the SQL Worksheet (continued)

6. **SQL History:** This displays a dialog box with information about the SQL statements that you have executed.
7. **Execute Explain Plan:** This generates the execution plan, which you can see by clicking the Explain tab.
8. **Autotrace:** This displays trace-related information when you execute the SQL statement by clicking the Autotrace icon. This information can help you to identify the SQL statements that will benefit from tuning.
9. **Clear:** This erases the statement or statements in the Enter SQL Statement box. Alternatively, press and hold [Ctrl] + [D] to erase the statements.

Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.



ORACLE

Executing SQL Statements

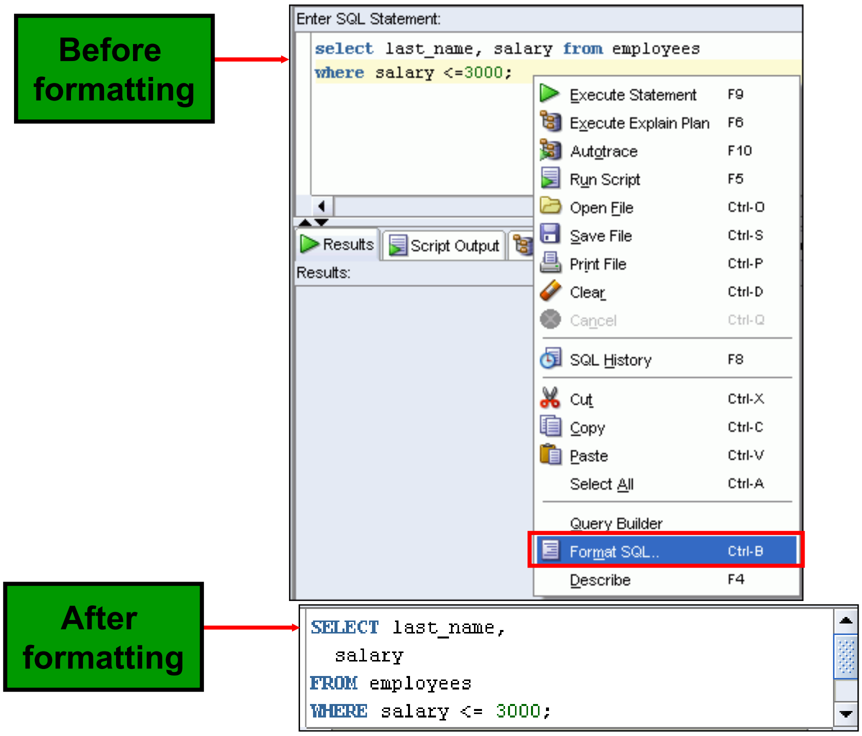
In the SQL Worksheet, you can use the Enter SQL Statement box to enter a single or multiple SQL statements. For a single statement, the semicolon at the end is optional.

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

To execute multiple SQL statements and see the results, click the Run Script icon. Alternatively, you can press [F5].

The example in the slide shows the difference in output for the same query when F9 key or Execute Statement is used versus the output when F5 or Run Script is used.

Formatting the SQL Code



ORACLE

I - 45 Copyright © 2007, Oracle. All rights reserved.

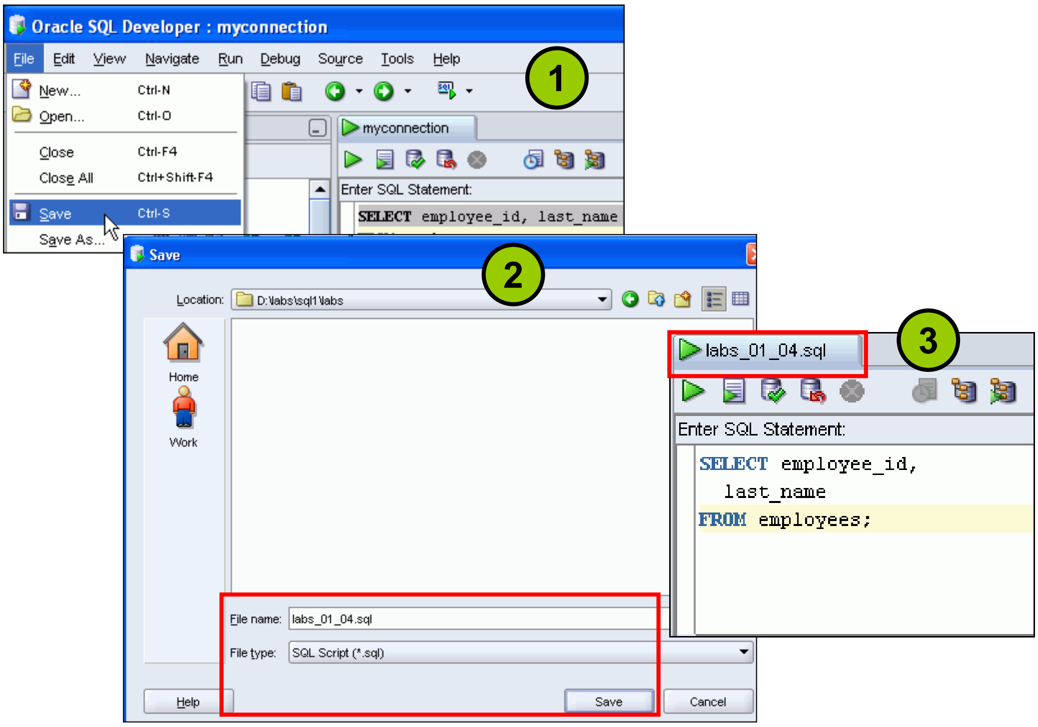
Formatting the SQL Code

You may want to beautify the indentation, spacing, capitalization, and the line separation of the SQL code. Oracle SQL Developer has the feature for formatting the SQL code.

To format the SQL code, right-click in the statement area, and select Format SQL.

In the example in the slide, before formatting, the keywords are not capitalized and the statement is not properly indented in the SQL code. After formatting, the SQL code is beautified with the keywords capitalized and the statement properly indented.

Saving SQL Statements

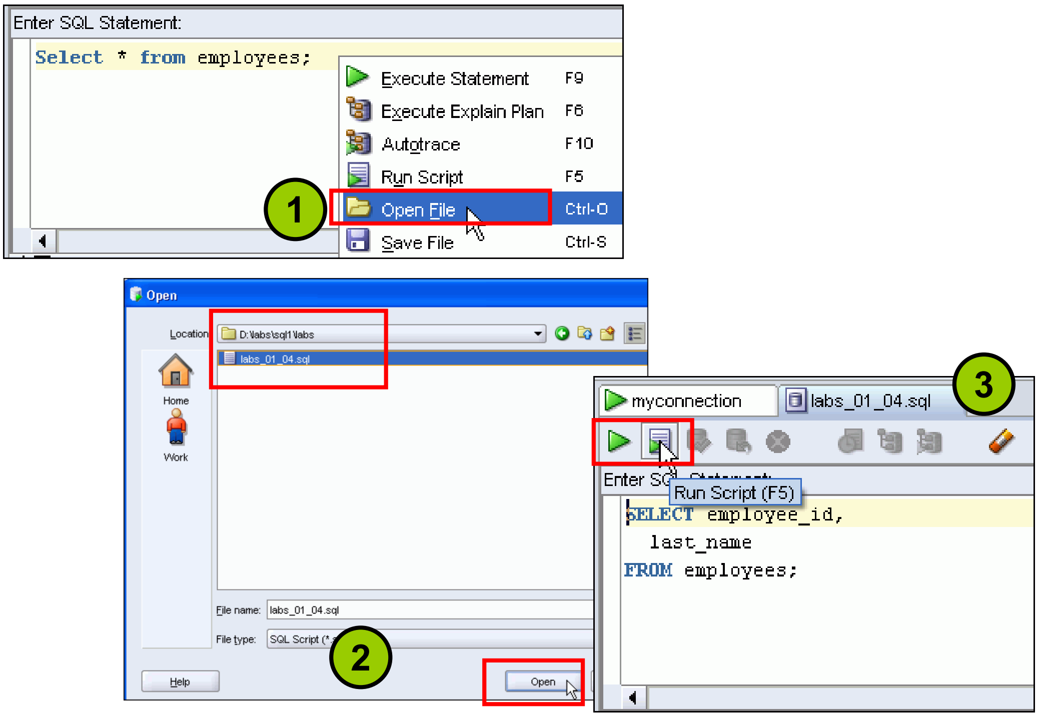


Saving SQL Statements

In most of the practices that you will perform, you will need to save a particular query in the SQL Worksheet as a `.sql` file. To do so, perform the following:

1. From the File menu, select Save or Save As (if you are renaming a current `.sql` script) or you can right-click the SQL Worksheet and select Save File. Alternatively, you can press and hold [CTRL] + [S].
2. In the Save dialog box, enter the appropriate filename. Make sure the extension is `.sql` or the File type is selected as SQL Script (*.sql). Click Save.
   Note: For this course, you need to save your `sql` scripts in the `D:\labs\sql1\labs` folder.
3. The SQL Worksheet is renamed to the filename that you saved the script as. Make sure you do not enter any other SQL statements in the same worksheet. To continue with other SQL queries, open a new worksheet.

Running Script Files



Running Script Files

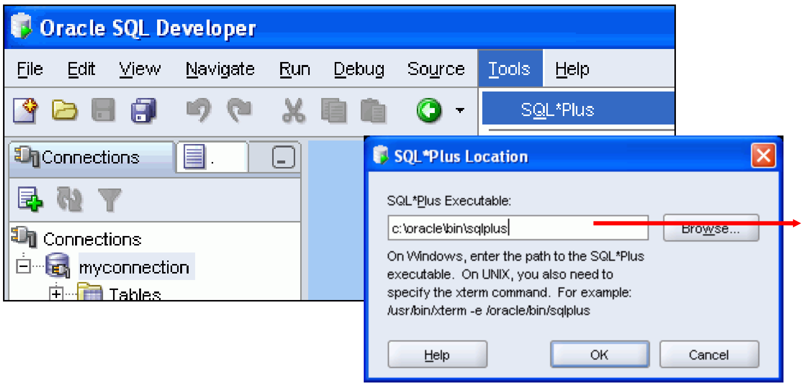
To run the saved `.sql` script files, perform the following:

1. Right-click the SQL Worksheet and select Open File, or select Open from the File menu. Alternatively, you can press and hold [CTRL] + [O].
2. In the Open dialog box, move to the `D:\labs\sql\labs` folder, or to the location in which you saved the script file, select the file and click Open.
3. The script file opens in a new worksheet. Now, you can run the script by either clicking the Execute Statement icon or the Run Script icon. Again, make sure you do not enter any other SQL statements in the same worksheet. To continue with other SQL queries, open a new worksheet.
   Note: You may want to set the default directory to `D:\labs\sql1` folder, so that every time you try to open or save a script, SQL Developer chooses the same path to look for scripts. From Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters. In the right pane, click Browse to set the default path to look for scripts and click OK.

Note: For more details on how to use the Oracle SQL Developer GUI interface for other data objects creation and data retrieval tasks, refer to Appendix G “Performing DML and DDL Operations Using the Oracle SQL Developer GUI.”

Starting SQL*Plus from Oracle SQL Developer

You can invoke the SQL*Plus command-line interface from Oracle SQL Developer.



Provide the location of the `sqlplus.exe` file only for the first time you invoke SQL*Plus.

Starting SQL*Plus from Oracle SQL Developer

The SQL Worksheet supports most of the SQL*Plus statements. SQL*Plus statements must be interpreted by the SQL Worksheet before being passed to the database; any SQL*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database. To display the SQL*Plus command window, from the Tools menu, select SQL*Plus. To use this feature, the system on which you are using Oracle SQL Developer must have an Oracle Home directory or folder, with a SQL*Plus executable in that location. If the location of the SQL*Plus executable is not already stored in your Oracle SQL Developer preferences, you are asked to specify its location.

Note: If the Tools > SQL*Plus menu option is disabled, click the database connection, such as myconnection, in the Connections Navigator. The menu option is disabled if any SQL Worksheet is active.

For example, some of the SQL*Plus statements that are not supported by SQL Worksheet are:

- `append`
- `archive`
- `attribute`
- `break`

For a complete list of the SQL*Plus statements that are supported and not supported by the SQL Worksheet, refer to the *SQL*Plus Statements Supported and Not Supported in SQL Worksheet* topic in the Oracle SQL Developer online Help.

SQL Statements in SQL*Plus

In Oracle Database 11*g*, SQL*Plus is a command-line interface.

I - 49

SQL Statements in SQL*Plus

Oracle SQL*Plus is a command-line interface with which you can submit SQL statements and PL/SQL blocks for execution and receive the results in an application or command window.

SQL*Plus is:

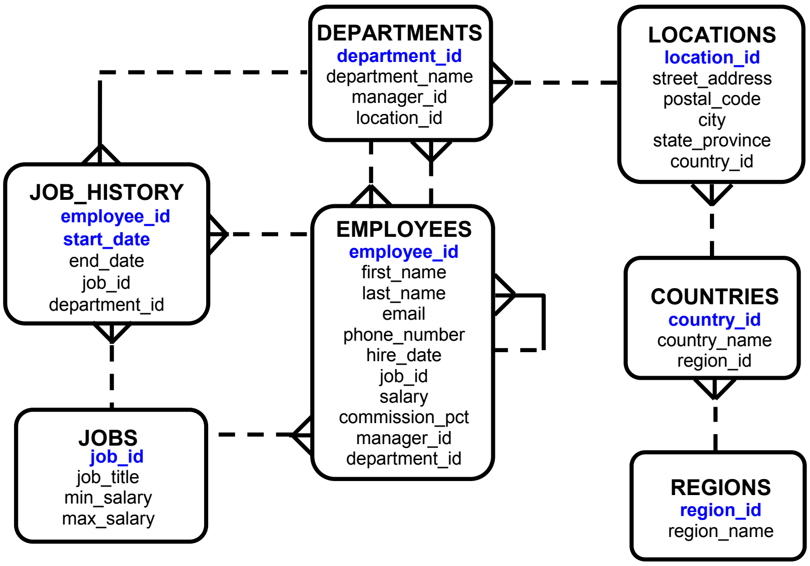
- Shipped with the database
- Installed on a client and on the database server system
- Accessed through from an icon or the command line

Note: If you do not have access to Oracle SQL Developer and would prefer to use SQL* Plus, the classroom setup provides SQL*Plus icon on your desktop. It may also be useful to use it in cases where Oracle SQL Developer does not support any SQL* Plus command.

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11*g* and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- Overview of Oracle SQL Developer
- The HR schema and the tables used in this course
- Oracle Database 11*g* documentation and additional resources

The `Human Resources (HR)` Schema



I - 51

The `Human Resources (HR)` Schema Description

The Human Resources (HR) schema is a part of the Oracle Sample Schemas that can be installed in an Oracle database. The practice sessions in this course use data from the HR schema.

Table Descriptions

- `REGIONS` contains rows that represent a region such as America, Asia, and so on.
- `COUNTRIES` contains rows for countries, each of which is associated with a region.
- `LOCATIONS` contains the specific address of a specific office, warehouse, or production site of a company in a particular country.
- `DEPARTMENTS` shows details about the departments in which the employees work. Each department may have a relationship representing the department manager in the `EMPLOYEES` table.
- `EMPLOYEES` contains details about each employee working for a department. Some employees may not be assigned to any department.
- `JOBS` contains the job types that can be held by each employee.
- `JOB_HISTORY` contains the job history of the employees. If an employee changes departments within a job or changes jobs within a department, then a new row is inserted into this table with the earlier job information of the employee.

Tables Used in the Course

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	9000	(null)	60	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	6000	(null)	60	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	4200	(null)	60	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	5800	(null)	50	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	3500	(null)	50	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	3100	(null)	50	CDAVIES	650.121.2994	29-JAN-97
					50	RMATOS	650.121.2874	15-MAR-98

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting	(null)	1700

DEPARTMENTS

GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

JOB_GRADES

Tables Used in the Course

The following main tables are used in this course:

- `EMPLOYEES` table: Gives details of all the employees
- `DEPARTMENTS` table: Gives details of all the departments
- `JOB_GRADES` table: Gives details of salaries for various grades

Apart from these tables, you will also use the other tables listed in the previous slide such as the `LOCATIONS` and the `JOB_HISTORY` table.

Note: The structure and data for all the tables are provided in Appendix B.

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11*g* and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- Overview of Oracle SQL Developer
- The HR schema and the tables used in this course
- Oracle Database 11*g* documentation and additional resources

Oracle Database 11*g* Documentation

- *Oracle Database New Features Guide 11g, Release 1 (11.1)*
- *Oracle Database Reference 11g, Release 1 (11.1)*
- *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*
- *Oracle Database Concepts 11g, Release 1 (11.1)*
- *Oracle Database SQL Developer User's Guide, Release 1.2*

ORACLE

I - 54

Oracle Database 11*g* Documentation

Navigate to http://www.oracle.com/pls/db111/homepage to access the Oracle Database 11*g* documentation library.

Additional Resources

For additional information about the Oracle Database 11*g*, refer to the following:

- *Oracle Database 11g: New Features eStudies*
- *Oracle by Example series (OBE): Oracle Database 11g*
  - http://www.oracle.com/technology/obe/11gr1_db/index.htm

Summary

- Oracle Database 11*g* extends:
  - the benefits of infrastructure grids
  - the existing information management capabilities
  - the capabilities to use the major application development environments such as PL/SQL, Java/JDBC, .NET, XML, and so on
- The database is based on ORDBMS.
- Relational databases are composed of relations, managed by relational operations, and governed by data integrity constraints.
- With the Oracle server, you can store and manage information by using SQL.

Summary

Relational database management systems are composed of objects or relations. They are managed by operations and governed by data integrity constraints.

Oracle Corporation produces products and services to meet your RDBMS needs. The main products are the following:

- Oracle Database 11*g* with which you store and manage information by using SQL
- Oracle Fusion Middleware with which you develop, deploy, and manage modular business services that can be integrated and reused
- Oracle Enterprise Manager Grid Control 10*g*, which you use to manage and automate administrative tasks across sets of systems in a grid environment

SQL

The Oracle server supports ANSI-standard SQL and contains extensions. SQL is the language that is used to communicate with the server to access, manipulate, and control data.

Practice I: Overview

This practice covers the following topics:

- Running the Oracle SQL Developer demo
- Starting Oracle SQL Developer, creating a new database connection, and browsing the HR tables

I - 57

Practice I: Overview

In this practice, you perform the following:

- Run through the Oracle SQL Developer demo.
- Use Oracle SQL Developer to examine data objects in the `ORA` account assigned to you. The `ORA` accounts contain the HR schema tables.

Note the following location for the lab files:

`D:\labs\SQL1\labs`

If you are asked to save any lab files, save them in this location.

In any practice, there maybe exercises that are prefaced with the phrases "If you have time" or "If you want an extra challenge." Work on these exercises only if you have completed all other exercises within the allocated time and would like a further challenge to your skills.

Perform the practices slowly and precisely. You can experiment with saving and running command files. If you have any questions at any time, ask your instructor.

Note: All written practices use Oracle SQL Developer as the development environment. Although it is recommended that you use Oracle SQL Developer, you can also use SQL*Plus that is available in this course.

Practice I

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

Run Through the Oracle SQL Developer Demo: Creating a Database Connection

1. Access the demo"Creating a database connection" at:
   http://st-curriculum.oracle.com/tutorial/SQLDeveloper/html/module2/mod02_cp_newdbconn.htm

Starting Oracle SQL Developer

2. Start Oracle SQL Developer using the sqldeveloper desktop icon.

   Note: When you start SQL Developer for the first time, you need to provide the path to the `java.exe` file. This is already done for you as a part of the classroom setup. In any case, if you are prompted, enter the following path:
   `D:\app\Administrator\product\11.1.0\client_1\jdevstudio\jdk\bin`

Creating a New Oracle SQL Developer Database Connection

3. To create a new database connection, in the Connections Navigator, right-click Connections. Select New Connection from the menu. The New/Select Database Connection dialog box appears.
4. Create a database connection using the following information:
   a. Connection Name: `myconnection`.
   b. Username: `oraxx` where `xx` is the number of your PC (Ask your instructor to assign you one `ora` account out of the `ora1`-`ora20` range of accounts.).
   c. Password: `oraxx`
   d. Hostname: Enter the host name of the machine where your database server is running.
   e. Port: 1521
   f. SID: ORCL
   g. Ensure you select the Save Password check box.

Practice I (continued)

Testing and Connecting Using the Oracle SQL Developer Database Connection

5. Test the new connection.
6. If the status is Success, connect to the database using this new connection.

Browsing the Tables in the Connections Navigator

7. In the Connections Navigator, view the objects available to you in the Tables node. Verify that the following tables are present:

```
COUNTRIES
DEPARTMENTS
EMPLOYEES
JOB_GRADES
JOB_HISTORY
JOBS
LOCATIONS
REGIONS
```

8. Browse the structure of the `EMPLOYEES` table.
9. View the data of the `DEPARTMENTS` table.

Opening a SQL Worksheet

10. Open a new SQL Worksheet. Examine the shortcut icons available for the SQL Worksheet.

1

Retrieving Data Using the SQL `SELECT` Statement

ORACLE

Objectives

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

- List the capabilities of SQL `SELECT` statements
- Execute a basic `SELECT` statement

1 - 2

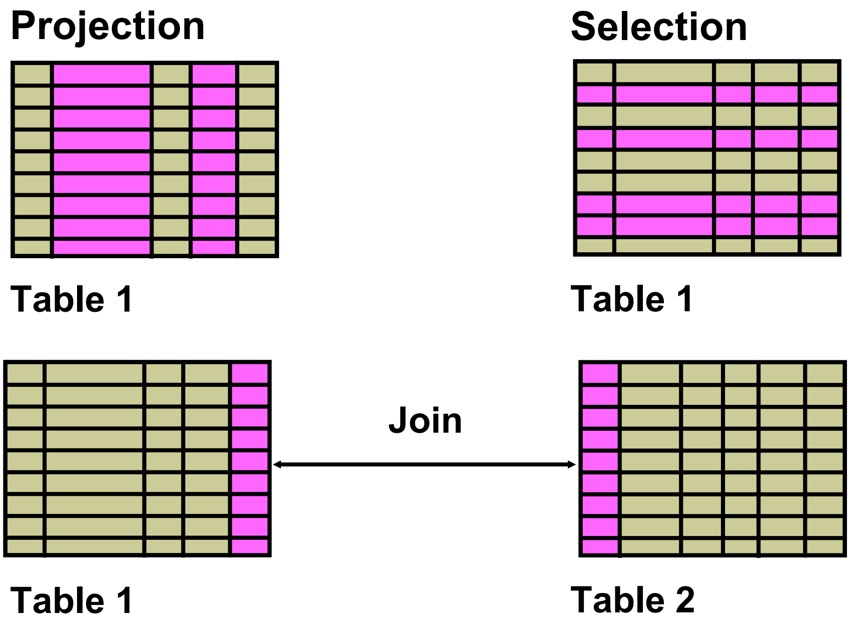
Objectives

To extract data from the database, you need to use the SQL `SELECT` statement. However, you may need to restrict the columns that are displayed. This lesson describes all the SQL statements that are needed to perform these actions. Further, you may want to create `SELECT` statements that can be used more than once.

Lesson Agenda

- Basic SELECT statement
- Arithmetic expressions and NULL values in the SELECT statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the DISTINCT keyword
- DESCRIBE command

Capabilities of SQL `SELECT` Statements



Capabilities of SQL `SELECT` Statements

A `SELECT` statement retrieves information from the database. With a `SELECT` statement, you can use the following capabilities:

- **Projection:** Select the columns in a table that are returned by a query. Select as few or as many of the columns as required.
- **Selection:** Select the rows in a table that are returned by a query. Various criteria can be used to restrict the rows that are retrieved.
- **Joining:** Bring together data that is stored in different tables by specifying the link between them. SQL joins are covered in more detail in the lesson titled "Displaying Data from Multiple Tables."

Basic SELECT Statement

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table;
```

- SELECT identifies the columns to be displayed.
- FROM identifies the table containing those columns.

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 identifies the table containing the columns that are listed in the SELECT clause

In the syntax:

SELECT	is a list of one or more columns
*	selects all columns
DISTINCT	suppresses duplicates
column|expression	selects the named column or the expression
alias	gives the selected columns different headings
FROM *table*	specifies the table containing the columns

Note: Throughout this course, the words *keyword*, *clause*, and *statement* are used as follows:

- A *keyword* refers to an individual SQL element.
  For example, SELECT and FROM are keywords.
- A *clause* is a part of a SQL statement.
  For example, SELECT employee_id, last_name, and so on is a clause.
- A *statement* is a combination of two or more clauses.
  For example, SELECT * FROM employees is a SQL statement.

Selecting All Columns

```
SELECT *
FROM   departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

Selecting All Columns

You can display all columns of data in a table by following the SELECT keyword with an asterisk (*). In the example in the slide, the department table contains four columns: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. The table contains eight rows, one for each department.

You can also display all columns in the table by listing all the columns after the SELECT keyword. For example, the following SQL statement (like the example in the slide) displays all columns and all rows of the DEPARTMENTS table:

```
SELECT   department_id, department_name, manager_id, location_id
FROM     departments;
```

Note: In SQL Developer, you can enter your SQL statement in a SQL Worksheet and click the "Execute Statement" icon or press [F9] to execute the statement. The output displayed in the Results tabbed page appears as shown in the slide.

Selecting Specific Columns

```
SELECT department_id, location_id
FROM   departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

Selecting Specific Columns

You can use the SELECT statement to display specific columns of the table by specifying the column names, separated by commas. The example in the slide displays all the department numbers and location numbers from the DEPARTMENTS table.

In the SELECT clause, specify the columns that you want in the order in which you want them to appear in the output. For example, to display location before department number (from left to right), you use the following statement:

```
SELECT location_id, department_id
FROM   departments;
```

	LOCATION_ID	DEPARTMENT_ID
1	1700	10
2	1800	20
3	1500	50
4	1400	60

...

Writing SQL Statements

- SQL statements are not case-sensitive.
- SQL statements can be entered 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.
- In SQL Developer, SQL statements can optionally be terminated by a semicolon (;). Semicolons are required when you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

1 - 8

Writing SQL Statements

By using the following simple rules and guidelines, you can construct valid statements that are both easy to read and edit:

- SQL statements are not case-sensitive (unless indicated).
- SQL statements can be entered on one or many lines.
- Keywords cannot be split across lines or abbreviated.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Indents should be used to make code more readable.
- Keywords typically are entered in uppercase; all other words, such as table names and columns names are entered in lowercase.

Executing SQL Statements

In SQL Developer, click the Run Script icon or press [F5] to run the command or commands in the SQL Worksheet. You can also click the Execute Statement icon or press [F9] to run a SQL statement in the SQL Worksheet. The Execute Statement icon executes the statement at the mouse pointer in the Enter SQL Statement box while the Run Script icon executes all the statements in the Enter SQL Statement box. The Execute Statement icon displays the output of the query on the Results tabbed page while the Run Script icon emulates the SQL*Plus display and shows the output on the Script Output tabbed page.

In SQL*Plus, terminate the SQL statement with a semicolon, and then press [Enter] to run the command.

Column Heading Defaults

- SQL Developer:
  - Default heading alignment: Left-aligned
  - Default heading display: Uppercase
- SQL*Plus:
  - Character and Date column headings are left-aligned.
  - Number column headings are right-aligned.
  - Default heading display: Uppercase

Column Heading Defaults

In SQL Developer, column headings are displayed in uppercase and are left-aligned.

```
SELECT last_name, hire_date, salary
FROM   employees;
```

	LAST_NAME	HIRE_DATE	SALARY
1	King	17-JUN-87	24000
2	Kochhar	21-SEP-89	17000
3	De Haan	13-JAN-93	17000
4	Hunold	03-JAN-90	9000
5	Ernst	21-MAY-91	6000
6	Lorentz	07-FEB-99	4200
7	Mourgos	16-NOV-99	5800
8	Rajs	17-OCT-95	3500

...

You can override the column heading display with an alias. Column aliases are covered later in this lesson.

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column Aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

ORACLE

Arithmetic Expressions

You may need to modify the way in which data is displayed, or you may want to perform calculations, or look at what-if scenarios. All these are possible using arithmetic expressions. An arithmetic expression can contain column names, constant numeric values, and the arithmetic operators.

Arithmetic Operators

The slide lists the arithmetic operators that are available in SQL. You can use arithmetic operators in any clause of a SQL statement (except the `FROM` clause).

Note: With the `DATE` and `TIMESTAMP` data types, you can use the addition and subtraction operators only.

Using Arithmetic Operators

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

...

ORACLE

1 - 12

Using Arithmetic Operators

The example in the slide uses the addition operator to calculate a salary increase of $300 for all employees. The slide also displays a SALARY+300 column in the output.

Note that the resultant calculated column, SALARY+300, is not a new column in the EMPLOYEES table; it is for display only. By default, the name of a new column comes from the calculation that generated it—in this case, salary+300.

Note: The Oracle server ignores blank spaces before and after the arithmetic operator.

Operator Precedence

If an arithmetic expression contains more than one operator, multiplication and division are evaluated first. If operators in an expression are of the same priority, then evaluation is done from left to right.

You can use parentheses to force the expression that is enclosed by the parentheses to be evaluated first.

Rules of Precedence:

- Multiplication and division occur before addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to override the default precedence or to clarify the statement.

Operator Precedence

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100

...

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200

...

1 - 13

Operator Precedence (continued)

The first example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation by multiplying the monthly salary with 12, plus a one-time bonus of $100. Note that multiplication is performed before addition.

Note: Use parentheses to reinforce the standard order of precedence and to improve clarity. For example, the expression in the slide can be written as `(12*salary)+100` with no change in the result.

Using Parentheses

You can override the rules of precedence by using parentheses to specify the desired order in which the operators are to be executed.

The second example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation as follows: adding a monthly bonus of $100 to the monthly salary, and then multiplying that subtotal with 12. Because of the parentheses, addition takes priority over multiplication.

Defining a Null Value

- Null is a value that is unavailable, unassigned, unknown, or inapplicable.
- Null is not the same as zero or a blank space.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
…				
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
…				
19	Higgins	AC_MGR	12000	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

ORACLE

Defining a Null Value

If a row lacks a data value for a particular column, that value is said to be *null* or to contain a null.

Null is a value that is unavailable, unassigned, unknown, or inapplicable. Null is not the same as zero or a blank space. Zero is a number and blank 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.

Note: By default, SQL Developer uses the literal, (null), to identify null values. However, you can set it to something more relevant to you. To do so, select Preferences from the Tools menu. In the Preferences dialog box, expand the Database node. Click Advanced Parameters and on the right pane, for the “Display Null value As,” enter the appropriate value.

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

	LAST_NAME	12*SALARY*COMMISSION_PCT
1	King	(null)
2	Kochhar	(null)

...

12	Zlotkey	25200
13	Abel	39600
14	Taylor	20640

...

19	Higgins	(null)
20	Gietz	(null)

ORACLE

1 - 15

Null Values in Arithmetic Expressions

If any column value in an arithmetic expression is null, the result is null. For example, if you attempt to perform division by zero, you get an error. However, if you divide a number by null, the result is a null or unknown.

In the example in the slide, employee King does not get any commission. Because the `COMMISSION_PCT` column in the arithmetic expression is null, the result is null.

For more information, see the section on “Basic Elements of Oracle SQL” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

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 if it is case-sensitive

Defining a Column Alias

When displaying the result of a query, SQL Developer normally uses the name of the selected column as the column heading. This heading may not be descriptive and, therefore, may be difficult to understand. You can change a column heading by using a column alias.

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

Using Column Aliases

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000

...

ORACLE

Using Column Aliases

The first example displays the names and the commission percentages of all the employees. Note that the optional `AS` keyword has been used before the column alias name. The result of the query is the same whether the `AS` keyword is used or not. Also, note that the SQL statement has the column aliases, `name` and `comm`, in lowercase, whereas the result of the query displays the column headings in uppercase. As mentioned in the previous slide, column headings appear in uppercase by default.

The second example displays the last names and annual salaries of all the employees. Because `Annual Salary` contains a space, it has been enclosed in double quotation marks. Note that the column heading in the output is exactly the same as the column alias.

Lesson Agenda

- Basic SELECT Statement
- Arithmetic Expressions and NULL values in SELECT statement
- Column Aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the DISTINCT keyword
- DESCRIBE command

1 - 19

Concatenation Operator

A concatenation operator:

- Links columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a resultant column that is a character expression

```
SELECT    last_name||job_id AS "Employees"
FROM      employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP

...

ORACLE

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.

In the example, `LAST_NAME` and `JOB_ID` are concatenated, and given the alias `Employees`. Note that the last name of the employee and the job code are combined to make a single output column.

The `AS` keyword before the alias name makes the `SELECT` clause easier to read.

Null Values with the Concatenation Operator

If you concatenate a null value with a character string, the result is a character string. `LAST_NAME || NULL` results in `LAST_NAME`.

Note: You can also concatenate date expressions with other expressions or columns.

Literal Character Strings

- A literal is a character, a number, or a date that is included in the `SELECT` statement.
- Date and character literal values must be enclosed within single quotation marks.
- Each character string is output once for each row returned.

1 - 21

Literal Character Strings

A literal is a character, a number, or a date that is included in the `SELECT` list. It is not a column name or a column alias. It is printed for each row returned. Literal strings of free-format text can be included in the query result and are treated the same as a column in the `SELECT` list.

Date and character literals *must* be enclosed within single quotation marks (`' '`); number literals need not be enclosed in a similar manner.

Using Literal Character Strings

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP

…

18	Vargas is a ST_CLERK
19	Whalen is a AD_ASST
20	Zlotkey is a SA_MAN

ORACLE

Using Literal Character Strings

The example in the slide displays the last names and job codes of all employees. The column has the heading Employee Details. Note the spaces between the single quotation marks in the `SELECT` statement. The spaces improve the readability of the output.

In the following example, the last name and salary for each employee are concatenated with a literal, to give the returned rows more meaning:

```
SELECT last_name ||': 1 Month salary = '||salary Monthly
FROM   employees;
```

	MONTHLY
1	King: 1 Month salary = 24000
2	Kochhar: 1 Month salary = 17000
3	De Haan: 1 Month salary = 17000
4	Hunold: 1 Month salary = 9000
5	Ernst: 1 Month salary = 6000
6	Lorentz: 1 Month salary = 4200
7	Mourgos: 1 Month salary = 5800
8	Rajs: 1 Month salary = 3500

…

Alternative Quote (q) Operator

- Specify your own quotation mark delimiter.
- Select any delimiter.
- Increase readability and usability.

```
SELECT department_name || ' Department' ||
       q'['s Manager Id: ]'
       || manager_id
       AS "Department and Manager"
FROM departments;
```

	Department and Manager
1	Administration Department's Manager Id:200
2	Marketing Department's Manager Id:201
3	Shipping Department's Manager Id:124
4	IT Department's Manager Id:103
5	Sales Department's Manager Id:149
6	Executive Department's Manager Id:100
7	Accounting Department's Manager Id:205
8	Contracting Department's Manager Id:

Alternative Quote (q) Operator

Many SQL statements use character literals in expressions or conditions. If the literal itself contains a single quotation mark, you can use the quote (q) operator and select your own quotation mark delimiter.

You can choose any convenient delimiter, single-byte or multibyte, or any of the following character pairs: [ ], { }, ( ), or < >.

In the example shown, the string contains a single quotation mark, which is normally interpreted as a delimiter of a character string. By using the q operator, however, brackets [] are used as the quotation mark delimiters. The string between the brackets delimiters is interpreted as a literal character string.

Duplicate Rows

The default display of queries is all rows, including duplicate rows.

```
SELECT department_id
FROM   employees;
```

1

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60

...

```
SELECT DISTINCT department_id
FROM   employees;
```

2

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110

...

Duplicate Rows

Unless you indicate otherwise, SQL displays the results of a query without eliminating the duplicate rows. The first example in the slide displays all the department numbers from the EMPLOYEES table. Note that the department numbers are repeated.

To eliminate duplicate rows in the result, include the DISTINCT keyword in the SELECT clause immediately after the SELECT keyword. In the second example in the slide, the EMPLOYEES table actually contains 20 rows, but there are only seven unique department numbers in the table.

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;
```

	DEPARTMENT_ID	JOB_ID
1	110	AC_ACCOUNT
2	90	AD_VP
3	50	ST_CLERK
4	80	SA_REP
5	50	ST_MAN

...

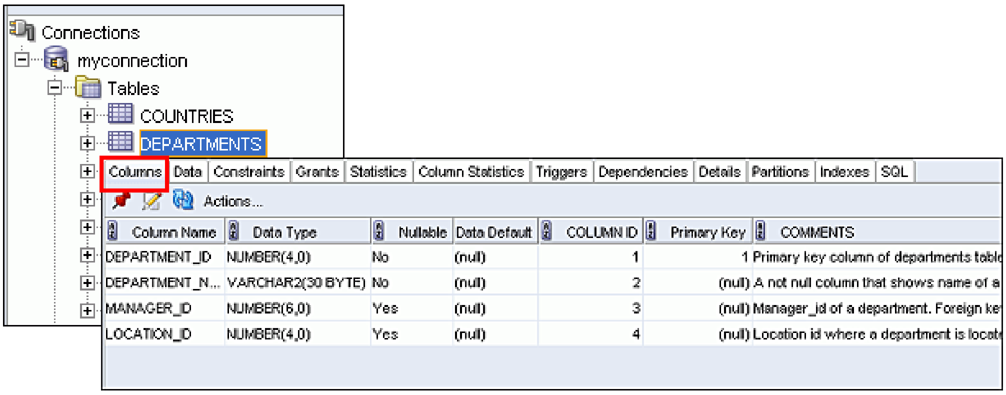
Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

Displaying the Table Structure

- Use the `DESCRIBE` command to display the structure of a table.
- Or, select the table in the Connections tree and use the Columns tab to view the table structure.

```
DESC[RIBE] tablename
```



1 - 26

Displaying the Table Structure

In SQL Developer, you can display the structure of a table by using the `DESCRIBE` command. The command displays the column names and the data types, and it shows you whether a column *must* contain data (that is, whether the column has a `NOT NULL` constraint).

In the syntax, *`table name`* is the name of any existing table, view, or synonym that is accessible to the user.

Using the SQL Developer GUI interface, you can select the table in the Connections tree and use the Columns tab to view the table structure.

Note: The `DESCRIBE` command is supported by both SQL*Plus and SQL Developer.

Using the DESCRIBE Command

```
DESCRIBE employees
```

```
DESCRIBE employees
Name                           Null     Type
------------------------------ -------- --------------------
EMPLOYEE_ID                    NOT NULL NUMBER(6)
FIRST_NAME                              VARCHAR2(20)
LAST_NAME                      NOT NULL VARCHAR2(25)
EMAIL                          NOT NULL VARCHAR2(25)
PHONE_NUMBER                            VARCHAR2(20)
HIRE_DATE                      NOT NULL DATE
JOB_ID                         NOT NULL VARCHAR2(10)
SALARY                                  NUMBER(8,2)
COMMISSION_PCT                          NUMBER(2,2)
MANAGER_ID                              NUMBER(6)
DEPARTMENT_ID                           NUMBER(4)

11 rows selected
```

ORACLE

Using the DESCRIBE Command

The example in the slide displays information about the structure of the EMPLOYEES table using the DESCRIBE command.

In the resulting display, *Null* indicates that the values for this column may be unknown. NOT NULL indicates that a column must contain data. *Type* displays the data type for a column.

The data types are described in the following table:

Data Type	Description
NUMBER(*p*, *s*)	Number value having a maximum number of digits *p*, with *s* digits to the right of the decimal point
VARCHAR2(*s*)	Variable-length character value of maximum size *s*
DATE	Date and time value between January 1, 4712 B.C. and December 31, A.D. 9999.
CHAR(*s*)	Fixed-length character value of size *s*

Summary

In this lesson, you should have learned how to:

- Write a `SELECT` statement that:
  - Returns all rows and columns from a table
  - Returns specified columns from a table
  - Uses column aliases to display more descriptive column headings

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table;
```


`SELECT` Statement

In this lesson, you should have learned how to retrieve data from a database table with the `SELECT` statement.

```
SELECT   *|{[DISTINCT] column [alias],...}
FROM     table;
```

In the syntax:

`SELECT`	is a list of one or more columns
`*`	selects all columns
`DISTINCT`	suppresses duplicates
`column|expression`	selects the named column or the expression
`alias`	gives the selected columns different headings
`FROM` *`table`*	specifies the table containing the columns

Practice 1: Overview

This practice covers the following topics:

- Selecting all data from different tables
- Describing the structure of tables
- Performing arithmetic calculations and specifying column names

Practice 1: Overview

In this practice, you write simple `SELECT` queries. The queries cover most of the `SELECT` clauses and operations that you learned in this lesson.

Practice 1

Part 1

Test your knowledge:

1. The following SELECT statement executes successfully:

```
SELECT last_name, job_id, salary AS Sal
FROM   employees;
```

   True/False

2. The following SELECT statement executes successfully:

```
SELECT *
FROM   job_grades;
```

   True/False

3. There are four coding errors in the following statement. Can you identify them?

```
SELECT    employee_id, last_name
sal x 12  ANNUAL SALARY
FROM      employees;
```

Part 2

Note the following points before you begin with the practices:

- Save all your lab files at the following location: `D:\labs\SQL1\labs`
- Enter your SQL statements in a SQL Worksheet. To save a script in SQL Developer, make sure the required SQL worksheet is active and then from the File menu, select Save As or right-click in the SQL Worksheet and select Save file to save your SQL statement as a lab_<lessonno>_<stepno>.sql script. When you are modifying an existing script, make sure you use Save As to save it with a different filename.
- To run the query, click the Execute Statement icon in the SQL Worksheet. Alternatively, you can press [F9]. For DML and DDL statements, use the Run Script icon or press [F5].
- After you have executed the query, make sure that you do not enter your next query in the same worksheet. Open a new worksheet.

You have been hired as a SQL programmer for Acme Corporation. Your first task is to create some reports based on data from the Human Resources tables.

4. Your first task is to determine the structure of the DEPARTMENTS table and its contents.

```
DESCRIBE departments
Name                           Null     Type
------------------------------ -------- ---------------
DEPARTMENT_ID                  NOT NULL NUMBER(4)
DEPARTMENT_NAME                NOT NULL VARCHAR2(30)
MANAGER_ID                              NUMBER(6)
LOCATION_ID                             NUMBER(4)

4 rows selected
```

Practice 1 (continued)

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

5. You need to determine the structure of the `EMPLOYEES` table.

```
DESCRIBE employees
Name                           Null     Type
------------------------------ -------- --------------------
EMPLOYEE_ID                    NOT NULL NUMBER(6)
FIRST_NAME                              VARCHAR2(20)
LAST_NAME                      NOT NULL VARCHAR2(25)
EMAIL                          NOT NULL VARCHAR2(25)
PHONE_NUMBER                            VARCHAR2(20)
HIRE_DATE                      NOT NULL DATE
JOB_ID                         NOT NULL VARCHAR2(10)
SALARY                                  NUMBER(8,2)
COMMISSION_PCT                          NUMBER(2,2)
MANAGER_ID                              NUMBER(6)
DEPARTMENT_ID                           NUMBER(4)

11 rows selected
```

The HR department wants a query to display the last name, job code, hire date, and employee number for each employee, with the employee number appearing first. Provide an alias `STARTDATE` for the `HIRE_DATE` column. Save your SQL statement to a file named `lab_01_05.sql` so that you can dispatch this file to the HR department.

Practice 1 (continued)

6. Test your query in the `lab_01_05.sql` file to ensure that it runs correctly.

   Note: After you have executed the query, make sure that you do not enter your next query in the same worksheet. Open a new worksheet.

	EMPLOYEE_ID	LAST_NAME	JOB_ID	STARTDATE
1	100	King	AD_PRES	17-JUN-87
2	101	Kochhar	AD_VP	21-SEP-89
3	102	De Haan	AD_VP	13-JAN-93
4	103	Hunold	IT_PROG	03-JAN-90
5	104	Ernst	IT_PROG	21-MAY-91
6	107	Lorentz	IT_PROG	07-FEB-99
7	124	Mourgos	ST_MAN	16-NOV-99
8	141	Rajs	ST_CLERK	17-OCT-95
9	142	Davies	ST_CLERK	29-JAN-97
10	143	Matos	ST_CLERK	15-MAR-98
...				
19	205	Higgins	AC_MGR	07-JUN-94
20	206	Gietz	AC_ACCOUNT	07-JUN-94

7. The HR department wants a query to display all unique job codes from the `EMPLOYEES` table.

	JOB_ID
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	IT_PROG
7	MK_MAN
8	MK_REP
9	SA_MAN
10	SA_REP
11	ST_CLERK
12	ST_MAN

Practice 1 (continued)

Part 3

If you have time, complete the following exercises:

8. The HR department wants more descriptive column headings for its report on employees. Copy the statement from `lab_01_05.sql` to a new SQL Worksheet. Name the column headings `Emp #`, `Employee`, `Job`, and `Hire Date`, respectively. Then run your query again.

	Emp #	Employee	Job	Hire Date
1	100	King	AD_PRES	17-JUN-87
2	101	Kochhar	AD_VP	21-SEP-89
3	102	De Haan	AD_VP	13-JAN-93
4	103	Hunold	IT_PROG	03-JAN-90
5	104	Ernst	IT_PROG	21-MAY-91
6	107	Lorentz	IT_PROG	07-FEB-99
7	124	Mourgos	ST_MAN	16-NOV-99
8	141	Rajs	ST_CLERK	17-OCT-95
9	142	Davies	ST_CLERK	29-JAN-97
10	143	Matos	ST_CLERK	15-MAR-98
...				
19	205	Higgins	AC_MGR	07-JUN-94
20	206	Gietz	AC_ACCOUNT	07-JUN-94

9. The HR department has requested a report of all employees and their job IDs. Display the last name concatenated with the job ID (separated by a comma and space) and name the column `Employee and Title`.

	Employee and Title
1	Abel, SA_REP
2	Davies, ST_CLERK
3	De Haan, AD_VP
4	Ernst, IT_PROG
5	Fay, MK_REP
6	Gietz, AC_ACCOUNT
7	Grant, SA_REP
8	Hartstein, MK_MAN
9	Higgins, AC_MGR
10	Hunold, IT_PROG
...	
19	Whalen, AD_ASST
20	Zlotkey, SA_MAN

Practice 1 (continued)

If you want an extra challenge, complete the following exercise:

10. To familiarize yourself with the data in the `EMPLOYEES` table, create a query to display all the data from that table. Separate each column output by a comma. Name the column title `THE_OUTPUT`.

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	THE_OUTPUT
1	100,Steven,King,SKING,515.123.4567,AD_PRES,,17-JUN-87,24000,,90
2	101,Neena,Kochhar,NKOCHHAR,515.123.4568,AD_VP,100,21-SEP-89,17000,,90
3	102,Lex,De Haan,LDEHAAN,515.123.4569,AD_VP,100,13-JAN-93,17000,,90
4	103,Alexander,Hunold,AHUNOLD,590.423.4567,IT_PROG,102,03-JAN-90,9000,,60
5	104,Bruce,Ernst,BERNST,590.423.4568,IT_PROG,103,21-MAY-91,6000,,60
6	107,Diana,Lorentz,DLORENTZ,590.423.5567,IT_PROG,103,07-FEB-99,4200,,60
7	124,Kevin,Mourgos,KMOURGOS,650.123.5234,ST_MAN,100,16-NOV-99,5800,,50
8	141,Trenna,Rajs,TRAJS,650.121.8009,ST_CLERK,124,17-OCT-95,3500,,50
9	142,Curtis,Davies,CDAVIES,650.121.2994,ST_CLERK,124,29-JAN-97,3100,,50
10	143,Randall,Matos,RMATOS,650.121.2874,ST_CLERK,124,15-MAR-98,2600,,50

...

19	205,Shelley,Higgins,SHIGGINS,515.123.8080,AC_MGR,101,07-JUN-94,12000,,110
20	206,William,Gietz,WGIETZ,515.123.8181,AC_ACCOUNT,205,07-JUN-94,8300,,110

Restricting and Sorting Data

Objectives

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

- Limit the rows that are retrieved by a query
- Sort the rows that are retrieved by a query
- Use ampersand substitution to restrict and sort output at run time

2 - 2

Objectives

When retrieving data from the database, you may need to do the following:

- Restrict the rows of data that are displayed
- Specify the order in which the rows are displayed

This lesson explains the SQL statements that you use to perform the actions listed above.

Lesson Agenda

- Limiting rows with:
  - The `WHERE` clause
  - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` conditions
  - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

Limiting Rows Using a Selection

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

...

"retrieve all employees in department 90"

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

ORACLE

Limiting Rows Using a Selection

In the example in the slide, assume that you want to display all the employees in department 90. The rows with a value of 90 in the `DEPARTMENT_ID` column are the only ones that are returned. This method of restriction is the basis of the `WHERE` clause in SQL.

Limiting the Rows that Are Selected

- Restrict the rows that are returned by using the `WHERE` clause:

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table
[WHERE condition(s)];
```

- The `WHERE` clause follows the `FROM` clause.

Limiting the Rows that Are Selected

You can restrict the rows that are 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.

In the syntax:

`WHERE`	restricts the query to rows that meet a condition
`condition`	is composed of column names, expressions, constants, and a comparison operator. A condition specifies a combination of one or more expressions and logical (Boolean) operators, and returns a value of `TRUE`, `FALSE`, or `UNKNOWN`.

The `WHERE` clause can compare values in columns, literal, 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 ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

ORACLE

2 - 6

Using the WHERE Clause

In the example, the `SELECT` statement retrieves the employee ID, last name, job ID, and department number of all employees who are in department 90.

Note: You cannot use column alias in the `WHERE` clause.

Character Strings and Dates

- Character strings and date values are enclosed with single quotation marks.
- Character values are case-sensitive and date values are format-sensitive.
- The default date display format is DD-MON-RR.

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen' ;
```

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-FEB-96' ;
```


Character Strings and Dates

Character strings and dates in the WHERE clause must be enclosed with single quotation marks (''). Number constants, however, should not be enclosed with single quotation marks.

All character searches are case-sensitive. In the following example, no rows are returned because the EMPLOYEES table stores all the last names in mixed case:

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'WHALEN';
```

Oracle databases store dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds. The default date display is in the DD-MON-RR format.

Note: For details about the RR format and about changing the default date format, see the lesson titled “Using Single-Row Functions to Customize Output.” Also, you learn about the use of single-row functions such as UPPER and LOWER to override the case sensitivity in the same lesson.

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
`BETWEEN ...AND...`	Between two values (inclusive)
`IN(set)`	Match any of a list of values
`LIKE`	Match a character pattern
`IS NULL`	Is a null value

Comparison Operators

Comparison operators are used in conditions that compare one expression to another value or expression. They are used in the `WHERE` clause in the following format:

Syntax

```
... WHERE expr operator value
```

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 symbols `!=` and `^=` can also represent the *not equal to* condition.

Using Comparison Operators

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

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

ORACLE

Using Comparison Operators

In the example, the SELECT statement retrieves the last name and salary from the EMPLOYEES table for any employee whose salary is less than or equal to $3,000. 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.

Range Conditions Using the BETWEEN Operator

Use the `BETWEEN` operator to display rows based on a range of values:

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Lower limit   Upper limit

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

Range Conditions Using the BETWEEN Operator

You can display rows based on a range of values using the `BETWEEN` operator. The range that you specify contains a lower limit and an upper limit.

The `SELECT` statement in the slide returns rows from the `EMPLOYEES` table for any employee whose salary is between $2,500 and $3,500.

Values that are specified with the `BETWEEN` operator are inclusive. However, you must specify the lower limit first.

You can also use the `BETWEEN` operator on character values:

```
SELECT last_name
FROM   employees
WHERE  last_name BETWEEN 'King' AND 'Smith';
```

	LAST_NAME
1	King
2	Kochhar
3	Lorentz
4	Matos
5	Mourgos
6	Rajs

Membership Condition Using the IN Operator

Use the `IN` operator to test for values in a list:

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

Membership Condition Using the IN Operator

To test for values in a specified set of values, use the `IN` operator. The condition defined using the `IN` operator is also known as the *membership condition*.

The slide example displays employee numbers, last names, salaries, and managers' employee numbers for all the employees whose manager's employee number is 100, 101, or 201.

The `IN` operator can be used with any data type. The following example returns a row from the `EMPLOYEES` table, for any employee whose last name is included in the list of names in the `WHERE` clause:

```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  last_name IN ('Hartstein', 'Vargas');
```

If characters or dates are used in the list, they must be enclosed with single quotation marks (`''`).

Note: The `IN` operator is internally evaluated by the Oracle server as a set of `OR` conditions, such as `a=value1` or `a=value2` or `a=value3`. Therefore, using the `IN` operator has no performance benefits and is used only for logical simplicity.

Pattern Matching Using the LIKE Operator

- Use the LIKE operator 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%' ;
```

2 - 12

Pattern Matching Using the LIKE Operator

You may not always know the exact value to search for. You can select rows that match a character pattern by using the LIKE operator. The character pattern–matching operation is referred to as a *wildcard* search. Two symbols can be used to construct the search string.

Symbol	Description
%	Represents any sequence of zero or more characters
_	Represents any single character

The SELECT statement in the slide returns the first name from the EMPLOYEES table for any employee whose first name begins with the letter "S." Note the uppercase "S." Consequently, names beginning with a lowercase "s" are not returned.

The LIKE operator can be used as a shortcut for some BETWEEN comparisons. The following example displays the last names and hire dates of all employees who joined between January, 1995 and December, 1995:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date LIKE '%95';
```

Combining Wildcard Characters

- You can combine the two wildcard characters (%, _) with literal characters for pattern matching:

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%' ;
```

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

- You can use the `ESCAPE` identifier to search for the actual `%` and _ symbols.

Combining Wildcard Characters

The `%` and _ symbols can be used in any combination with literal characters. The example in the slide displays the names of all employees whose last names have the letter “o” as the second character.

ESCAPE Identifier

When you need to have an exact match for the actual `%` and _ characters, use the `ESCAPE` identifier. This option specifies what the escape character is. If you want to search for strings that contain `SA_`, you can use the following SQL statement:

```
SELECT employee_id, last_name, job_id
FROM   employees WHERE  job_id LIKE '%SA\_%' ESCAPE '\';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	149	Zlotkey	SA_MAN
2	174	Abel	SA_REP
3	176	Taylor	SA_REP
4	178	Grant	SA_REP

The `ESCAPE` identifier identifies the backslash (\) as the escape character. In the SQL statement, the escape character precedes the underscore (_). This causes the Oracle server to interpret the underscore literally.

Using the NULL Conditions

Test for nulls with the IS NULL operator.

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL ;
```

	LAST_NAME	MANAGER_ID
1	King	(null)

Using the NULL Conditions

The NULL conditions include the IS NULL condition and the IS NOT NULL condition.

The IS NULL condition tests for nulls. A null value means that the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with =, because a null cannot be equal or unequal to any value. The slide example retrieves the last names and managers of all employees who do not have a manager.

Here is another example: To display the last name, job ID, and commission for all employees who are *not* entitled to receive a commission, use the following SQL statement:

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

	LAST_NAME	JOB_ID	COMMISSION_PCT
1	King	AD_PRES	(null)
2	Kochhar	AD_VP	(null)

...

15	Higgins	AC_MGR	(null)
16	Gietz	AC_ACCOUNT	(null)

Defining Conditions Using the Logical Operators

Operator	Meaning
AND	Returns TRUE if *both* component conditions are true
OR	Returns TRUE if *either* component condition is true
NOT	Returns TRUE if the condition is false

Defining Conditions Using the Logical Operators

A logical condition combines the result of two component conditions to produce a single result based on those conditions or it inverts the result of a single condition. A row is returned only if the overall result of the condition is true.

Three logical operators are available in SQL:

- AND
- OR
- NOT

All the examples so far have specified only one condition in the WHERE clause. You can use several conditions in a single WHERE clause using the AND and OR operators.

Using the AND Operator

AND requires both the component conditions to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
AND    job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

Using the AND Operator

In the example, both the component conditions must be true for any record to be selected. Therefore, only those employees who have a job title that contains the string 'MAN' *and* earn $10,000 or more are selected.

All character searches are case-sensitive, that is no rows are returned if 'MAN' is not uppercase. Further, character strings must be enclosed with quotation marks.

AND Truth Table

The following table shows the results of combining two expressions with AND:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Using the OR Operator

OR requires either component condition to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12000

ORACLE

2 - 17

Using the OR Operator

In the example, either component condition can be true for any record to be selected. Therefore, any employee who has a job ID that contains the string 'MAN' *or* earns $10,000 or more is selected.

OR Truth Table

The following table shows the results of combining two expressions with OR:

OR	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Using the NOT Operator

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

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

2 - 18

Using the NOT Operator

The slide example displays the last name and job ID of all employees whose job ID *is not* IT_PROG, ST_CLERK, or SA_REP.

NOT Truth Table

The following table shows the result of applying the NOT operator to a condition:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

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
```

Lesson Agenda

- Limiting rows with:
  - The WHERE clause
  - The comparison conditions using =, <=, BETWEEN, IN, LIKE, and NULL operators
  - Logical conditions using AND, OR, and NOT operators
- Rules of precedence for operators in an expression
- Sorting rows using the ORDER BY clause
- Substitution variables
- DEFINE and VERIFY commands

Rules of Precedence

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	`IS [NOT] NULL`, `LIKE`, `[NOT] IN`
5	`[NOT] BETWEEN`
6	Not equal to
7	`NOT` logical condition
8	`AND` logical condition
9	`OR` logical condition

You can use parentheses to override rules of precedence.

ORACLE

2 - 20

Rules of Precedence

The rules of precedence determine the order in which expressions are evaluated and calculated. The table in the slide lists the default order of precedence. However, you can override the default order by using parentheses around the expressions that you want to calculate first.

Rules of Precedence

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

1

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000
2	Abel	SA_REP	11000
3	Taylor	SA_REP	8600
4	Grant	SA_REP	7000

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

2

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000

ORACLE

Rules of Precedence (continued)

1. **Precedence of the AND Operator: Example**

In this example, there are two conditions:

- The first condition is that the job ID is AD_PRES *and* the salary is greater than $15,000.
- The second condition is that the job ID is SA_REP.

Therefore, 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."

2. **Using Parentheses: Example**

In this example, there are two conditions:

- The first condition is that the job ID is AD_PRES *or* SA_REP.
- The second condition is that the salary is greater than $15,000.

Therefore, 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."

Lesson Agenda

- Limiting rows with:
  - The WHERE clause
  - The comparison conditions using =, <=, BETWEEN, IN, LIKE, and NULL operators
  - Logical conditions using AND, OR, and NOT operators
- Rules of precedence for operators in an expression
- Sorting rows using the ORDER BY clause
- Substitution variables
- DEFINE and VERIFY commands

Using the ORDER BY Clause

- Sort retrieved rows with the ORDER BY clause:
  - ASC: Ascending order, default
  - DESC: Descending order
- The ORDER BY clause comes last in the SELECT statement:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES	90	17-JUN-87
2	Whalen	AD_ASST	10	17-SEP-87
3	Kochhar	AD_VP	90	21-SEP-89
4	Hunold	IT_PROG	60	03-JAN-90
5	Ernst	IT_PROG	60	21-MAY-91
6	De Haan	AD_VP	90	13-JAN-93

...

ORACLE

Using the ORDER BY Clause

The order of rows that are returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. However, if you use the ORDER BY clause, it must be the last clause of the SQL statement. Further, you can specify an expression, an alias, or a column position as the sort condition.

Syntax

```
SELECT          expr
FROM            table
[WHERE          condition(s)]
[ORDER BY  {column, expr, numeric_position} [ASC|DESC]];
```

In the syntax:

ORDER BY	specifies the order in which the retrieved rows are displayed
ASC	orders the rows in ascending order (this is the default order)
DESC	orders the rows in descending order

If the ORDER BY clause is not used, the sort order is undefined, and the Oracle server may not fetch rows in the same order for the same query twice. Use the ORDER BY clause to display the rows in a specific order.

Note: Use the keywords NULLS FIRST or NULLS LAST to specify whether returned rows containing null values should appear first or last in the ordering sequence.

Sorting

- Sorting in descending order:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC ;
```

1

- Sorting by column alias:

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```

2

Sorting

The default sort order is ascending:

- Numeric values are displayed with the lowest values first (for example, 1 to 999).
- Date values are displayed with the earliest value first (for example, 01-JAN-92 before 01-JAN-95).
- Character values are displayed in the alphabetical order (for example, "A" first and "Z" last).
- Null values are displayed last for ascending sequences and first for descending sequences.
- You can also sort by a column that is not in the `SELECT` list.

Examples:

1. To reverse the order in which the rows are displayed, specify the `DESC` keyword after the column name in the `ORDER BY` clause. The slide example sorts the result by the most recently hired employee.
2. You can also use a column alias in the `ORDER BY` clause. The slide example sorts the data by annual salary.

Sorting

- Sorting by using the column's numeric position:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY 3;
```

3

- Sorting by multiple columns:

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

4

Sorting (continued)

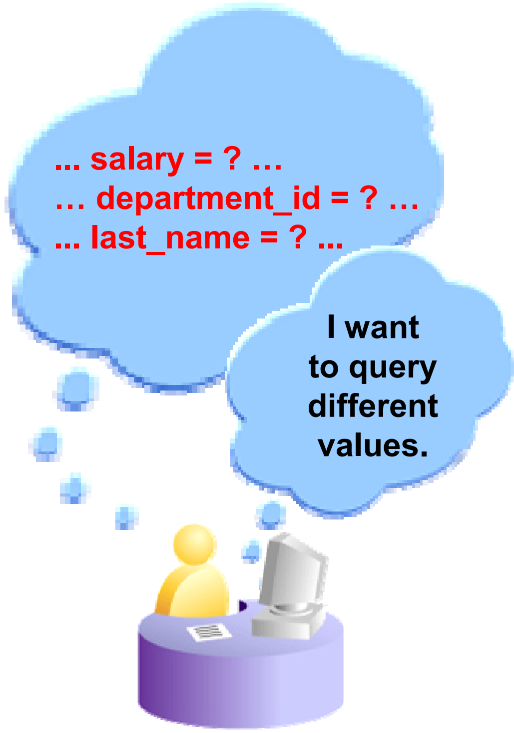
Examples:

3. You can sort query results by specifying the numeric position of the column in the `SELECT` clause. The slide example sorts the result by the `department_id` as this column is at the third position in the `SELECT` clause.
4. You can sort query results by more than one column. The sort limit is the number of columns in the given table. In the `ORDER BY` clause, specify the columns and separate the column names using commas. If you want to reverse the order of a column, specify `DESC` after its name.

Lesson Agenda

- Limiting rows with:
  - The `WHERE` clause
  - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
  - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

Substitution Variables



Substitution Variables

So far, all the SQL statements were executed with predetermined columns, conditions and their values. Suppose that you want a query that lists the employees with various jobs and not just those whose `job_ID` is `SA_REP`. You can edit the `WHERE` clause to provide a different value each time you run the command, but there is also an easier way.

By using a substitution variable in place of the exact values in the `WHERE` clause, you can run the same query for different values.

You can create reports that prompt users to supply their own values to restrict the range of data returned, by using substitution variables. You can embed *substitution variables* in a command file or in a single SQL statement. A variable can be thought of as a container in which values are temporarily stored. When the statement is run, the stored value is substituted.

Substitution Variables

- Use substitution variables to:
  - Temporarily store values with single-ampersand (`&`) and double-ampersand (`&&`) substitution
- Use substitution variables to supplement the following:
  - `WHERE` conditions
  - `ORDER BY` clauses
  - Column expressions
  - Table names
  - Entire `SELECT` statements

ORACLE

Substitution Variables (continued)

You can use single-ampersand (`&`) substitution variables to temporarily store values.

You can also predefine variables by using the `DEFINE` command. `DEFINE` creates and assigns a value to a variable.

Restricted Ranges of Data: Examples

- Reporting figures only for the current quarter or specified date range
- Reporting on data relevant only to the user requesting the report
- Displaying personnel only within a given department

Other Interactive Effects

Interactive effects are not restricted to direct user interaction with the `WHERE` clause. The same principles can also be used to achieve other goals, such as:

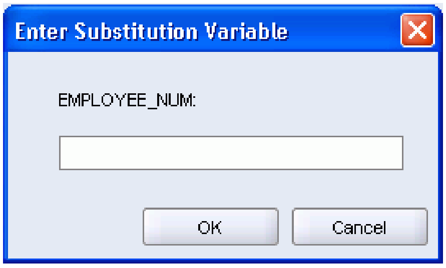
- Obtaining input values from a file rather than from a person
- Passing values from one SQL statement to another

Note: Both SQL Developer and SQL* Plus support the substitution variables and the `DEFINE/UNDEFINE` commands. Though SQL Developer or SQL* Plus does not support validation checks (except for data type) on user input.

Using the Single-Ampersand Substitution Variable

Use a variable prefixed with an ampersand (&) to prompt the user for a value:

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```



Using the Single-Ampersand Substitution Variable

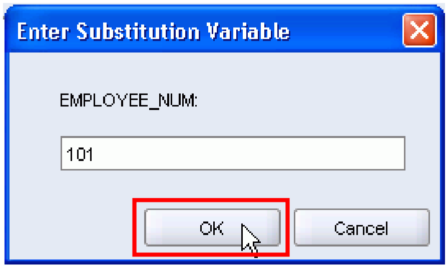
When running a report, users often want to restrict the data that is returned dynamically. SQL*Plus or SQL Developer provides this flexibility with user variables. Use an ampersand (&) to identify each variable in your SQL statement. However, you do not need to define the value of each variable.

Notation	Description
&user_variable	Indicates a variable in a SQL statement; if the variable does not exist, SQL*Plus or SQL Developer prompts the user for a value (the new variable is discarded after it is used.)

The example in the slide creates a SQL Developer substitution variable for an employee number. When the statement is executed, SQL Developer prompts the user for an employee number and then displays the employee number, last name, salary, and department number for that employee.

With the single ampersand, the user is prompted every time the command is executed if the variable does not exist.

Using the Single-Ampersand Substitution Variable



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

ORACLE

2 - 30

Using the Single-Ampersand Substitution Variable (continued)

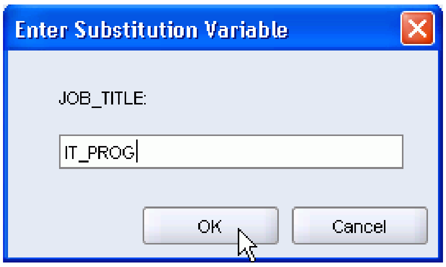
When SQL Developer detects that the SQL statement contains an ampersand, you are prompted to enter a value for the substitution variable that is named in the SQL statement.

After you enter a value and click the OK button, the results are displayed in the Results tab of your SQL Developer session.

Character and Date Values with Substitution Variables

Use single quotation marks for date and character values:

```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```



	LAST_NAME	DEPARTMENT_ID	SALARY*12
1	Hunold	60	108000
2	Ernst	60	72000
3	Lorentz	60	50400

Character and Date Values with Substitution Variables

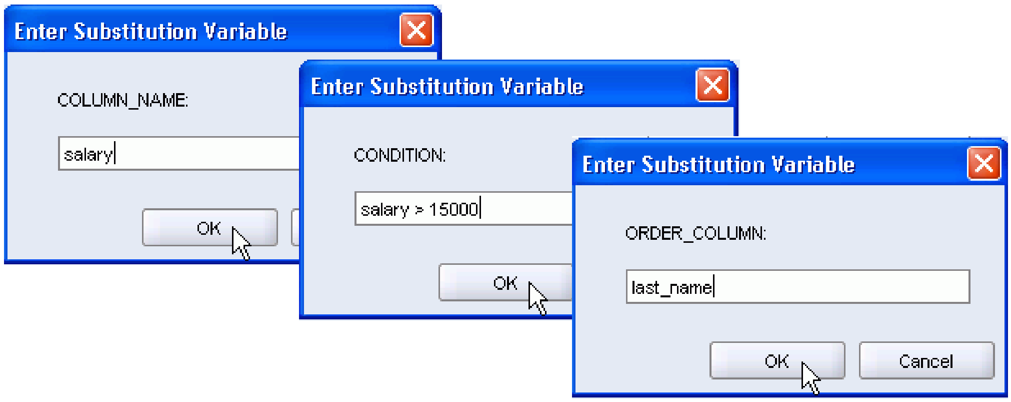
In a `WHERE` clause, date and character values must be enclosed with single quotation marks. The same rule applies to the substitution variables.

Enclose the variable with single quotation marks within the SQL statement itself.

The slide shows a query to retrieve the employee names, department numbers, and annual salaries of all employees based on the job title value of the SQL Developer substitution variable.

Specifying Column Names, Expressions, and Text

```
SELECT employee_id, last_name, job_id,&column_name
FROM   employees
WHERE  &condition
ORDER BY &order_column ;
```



Specifying Column Names, Expressions, and Text

You can use the substitution variables not only in the `WHERE` clause of a SQL statement, but also as substitution for column names, expressions, or text.

Example:

The slide example displays the employee number, last name, job title, and any other column that is specified by the user at run time, from the `EMPLOYEES` table. For each substitution variable in the `SELECT` statement, you are prompted to enter a value, and then click OK to proceed.

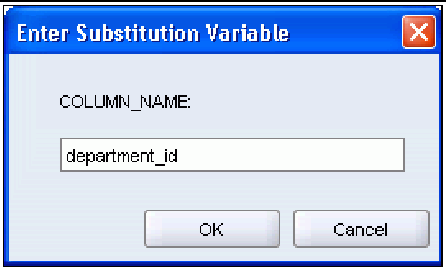
If you do not enter a value for the substitution variable, you get an error when you execute the preceding statement.

Note: A substitution variable can be used anywhere in the `SELECT` statement, except as the first word entered at the command prompt.

Using the Double-Ampersand Substitution Variable

Use double ampersand (`&&`) if you want to reuse the variable value without prompting the user each time:

```
SELECT   employee_id, last_name, job_id, &&column_name
FROM     employees
ORDER BY &column_name ;
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200	Whalen	AD_ASST	10
2	201	Hartstein	MK_MAN	20
3	202	Fay	MK_REP	20

...

Using the Double-Ampersand Substitution Variable

You can use the double-ampersand (`&&`) substitution variable if you want to reuse the variable value without prompting the user each time. The user sees the prompt for the value only once. In the example in the slide, the user is asked to give the value for the variable, `column_name`, only once. The value that is supplied by the user (`department_id`) is used for both display and ordering of data. If you run the query again, you will not be prompted for the value of the variable.

SQL Developer stores the value that is supplied by using the `DEFINE` command; it uses it again whenever you reference the variable name. After a user variable is in place, you need to use the `UNDEFINE` command to delete it:

```
UNDEFINE column_name
```

Lesson Agenda

- Limiting rows with:
  - The `WHERE` clause
  - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
  - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

Using the DEFINE Command

- Use the DEFINE command to create and assign a value to a variable.
- Use the UNDEFINE command to remove a variable.

```
DEFINE employee_num = 200

SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;

UNDEFINE employee_num
```


Using the DEFINE Command

The example shown creates a substitution variable for an employee number by using the DEFINE command. At run time, this displays the employee number, name, salary, and department number for that employee.

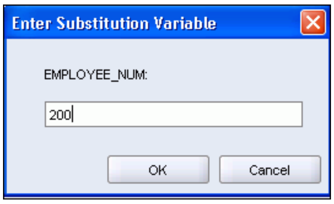
Because the variable is created using the SQL Developer DEFINE command, the user is not prompted to enter a value for the employee number. Instead, the defined variable value is automatically substituted in the SELECT statement.

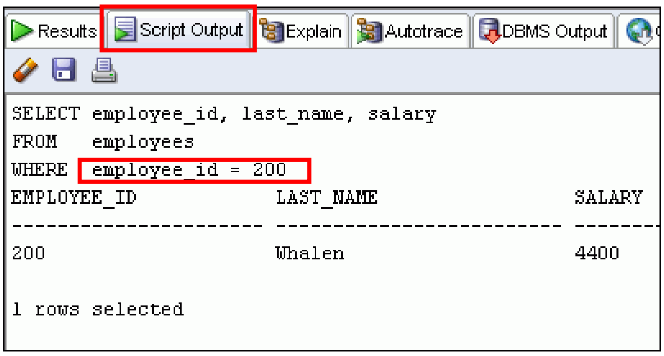
The EMPLOYEE_NUM substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

Using the `VERIFY` Command

Use the `VERIFY` command to toggle the display of the substitution variable, both before and after SQL Developer replaces substitution variables with values:

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM   employees
WHERE  employee_id = &employee_num;
```





Using the `VERIFY` Command

To confirm the changes in the SQL statement, use the `VERIFY` command. Setting `SET VERIFY ON` forces SQL Developer to display the text of a command after it replaces substitution variables with values. To see the `VERIFY` output, you should use the Run Script (F5) icon in the SQL Worksheet. SQL Developer displays the text of a command after it replaces substitution variables with values**,** in the Script Output tab as shown in the slide.

The example in the slide displays the new value of the `EMPLOYEE_ID` column in the SQL statement followed by the output.

SQL*Plus System Variables

SQL*Plus uses various system variables that control the working environment. One of the variables is `VERIFY`. To obtain a complete list of all the system variables, you can issue the `SHOW ALL` command on the SQL*Plus command prompt.

Summary

In this lesson, you should have learned how to:

- Use the `WHERE` clause to restrict rows of output:
  - Use the comparison conditions
  - Use the `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
  - Apply the logical `AND`, `OR`, and `NOT` operators
- Use the `ORDER BY` clause to sort rows of output:

```
SELECT   *|{[DISTINCT] column|expression [alias],...}
FROM     table
[WHERE   condition(s)]
[ORDER BY {column, expr, alias} [ASC|DESC]] ;
```

- Use ampersand substitution to restrict and sort output at run time

Summary

In this lesson, you should have learned about restricting and sorting rows that are returned by the `SELECT` statement. You should also have learned how to implement various operators and conditions.

By using the substitution variables, you can add flexibility to your SQL statements. This enables the queries to prompt for the filter condition for the rows during run time.

Practice 2: Overview

This practice covers the following topics:

- Selecting data and changing the order of the rows that are displayed
- Restricting rows by using the `WHERE` clause
- Sorting rows by using the `ORDER BY` clause
- Using substitution variables to add flexibility to your SQL `SELECT` statements

Practice 2: Overview

In this practice, you build more reports, including statements that use the `WHERE` clause and the `ORDER BY` clause. You make the SQL statements more reusable and generic by including the ampersand substitution.

Practice 2

The HR department needs your assistance in creating some queries.

1. Because of budget issues, the HR department needs a report that displays the last name and salary of employees who earn more than $12,000. Save your SQL statement as a file named `lab_02_01.sql`. Run your query.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000

2. Open a new SQL Worksheet. Create a report that displays the last name and department number for employee number 176. Run the query.

	LAST_NAME	DEPARTMENT_ID
1	Taylor	80

3. The HR department needs to find high-salary and low-salary employees. Modify `lab_02_01.sql` to display the last name and salary for any employee whose salary is not in the range of $5,000 to $12,000. Save your SQL statement as `lab_02_03.sql`.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Lorentz	4200
5	Rajs	3500
6	Davies	3100
7	Matos	2600
8	Vargas	2500
9	Whalen	4400
10	Hartstein	13000

Practice 2 (continued)

4. Create a report to display the last name, job ID, and start date for the employees with the last names of Matos and Taylor. Order the query in ascending order by the start date.

	LAST_NAME	JOB_ID	HIRE_DATE
1	Matos	ST_CLERK	15-MAR-98
2	Taylor	SA_REP	24-MAR-98

5. Display the last name and department number of all employees in departments 20 or 50 in ascending alphabetical order by name.

	LAST_NAME	DEPARTMENT_ID
1	Davies	50
2	Fay	20
3	Hartstein	20
4	Matos	50
5	Mourgos	50
6	Rajs	50
7	Vargas	50

6. Modify `lab_02_03.sql` to display the last name and salary of employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns `Employee` and `Monthly Salary`, respectively. Resave `lab_02_03.sql` as `lab_02_06.sql`. Run the statement in `lab_02_06.sql`.

	Employee	Monthly Salary
1	Fay	6000
2	Mourgos	5800

Practice 2 (continued)

7. The HR department needs a report that displays the last name and hire date for all employees who were hired in 1994.

	LAST_NAME	HIRE_DATE
1	Higgins	07-JUN-94
2	Gietz	07-JUN-94

8. Create a report to display the last name and job title of all employees who do not have a manager.

	LAST_NAME	JOB_ID
1	King	AD_PRES

9. Create a report to display the last name, salary, and commission of all employees who earn commissions. Sort data in descending order of salary and commissions. Use the column's numeric position in the `ORDER BY` clause.

	LAST_NAME	SALARY	COMMISSION_PCT
1	Abel	11000	0.3
2	Zlotkey	10500	0.2
3	Taylor	8600	0.2
4	Grant	7000	0.15

10. Members of the HR department want to have more flexibility with the queries that you are writing. They would like a report that displays the last name and salary of employees who earn more than an amount that the user specifies after a prompt. Save this query to a file named `lab_02_10.sql`. If you enter `12000` when prompted, the report displays the following results:

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000

Practice 2 (continued)

11. The HR department wants to run reports based on a manager. Create a query that prompts the user for a manager ID and generates the employee ID, last name, salary, and department for that manager's employees. The HR department wants the ability to sort the report on a selected column. You can test the data with the following values:

manager_id = 103, sorted by last_name:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	104	Ernst	6000	60
2	107	Lorentz	4200	60

manager_id = 201, sorted by salary:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	202	Fay	6000	20

manager_id = 124, sorted by employee_id:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	141	Rajs	3500	50
2	142	Davies	3100	50
3	143	Matos	2600	50
4	144	Vargas	2500	50

Practice 2 (continued)

If you have time, complete the following exercises:

12. Display all employee last names in which the third letter of the name is "a."

	LAST_NAME
1	Grant
2	Whalen

13. Display the last names of all employees who have both an "a" and an "e" in their last name.

	LAST_NAME
1	Davies
2	De Haan
3	Hartstein
4	Whalen

If you want an extra challenge, complete the following exercises:

14. Display the last name, job, and salary for all employees whose jobs are either those of a sales representative or of a stock clerk, and whose salaries are not equal to $2,500, $3,500, or $7,000.

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000
2	Taylor	SA_REP	8600
3	Davies	ST_CLERK	3100
4	Matos	ST_CLERK	2600

15. Modify `lab_02_06.sql` to display the last name, salary, and commission for all employees whose commission is 20%. Resave `lab_02_06.sql` as `lab_02_15.sql`. Rerun the statement in `lab_02_15.sql`.

	Employee	Monthly Salary	COMMISSION_PCT
1	Zlotkey	10500	0.2
2	Taylor	8600	0.2

Using Single-Row Functions to Customize Output

Objectives

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

- Describe various types of functions available in SQL
- Use character, number, and date functions in `SELECT` statements

Objectives

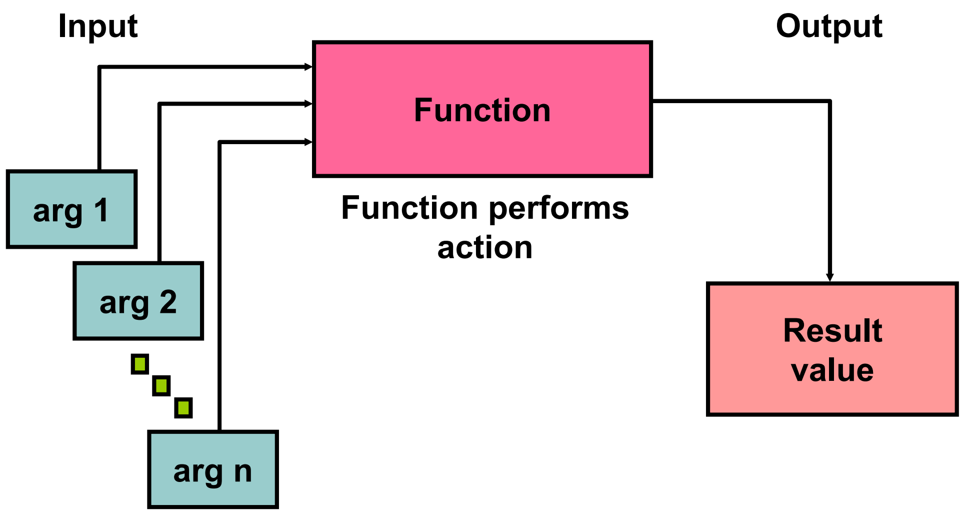
Functions make the basic query block more powerful, and they are used to manipulate data values. This is the first of two lessons that explore functions. It focuses on single-row character, number, and date functions.

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date functions

3 - 3

SQL Functions



SQL Functions

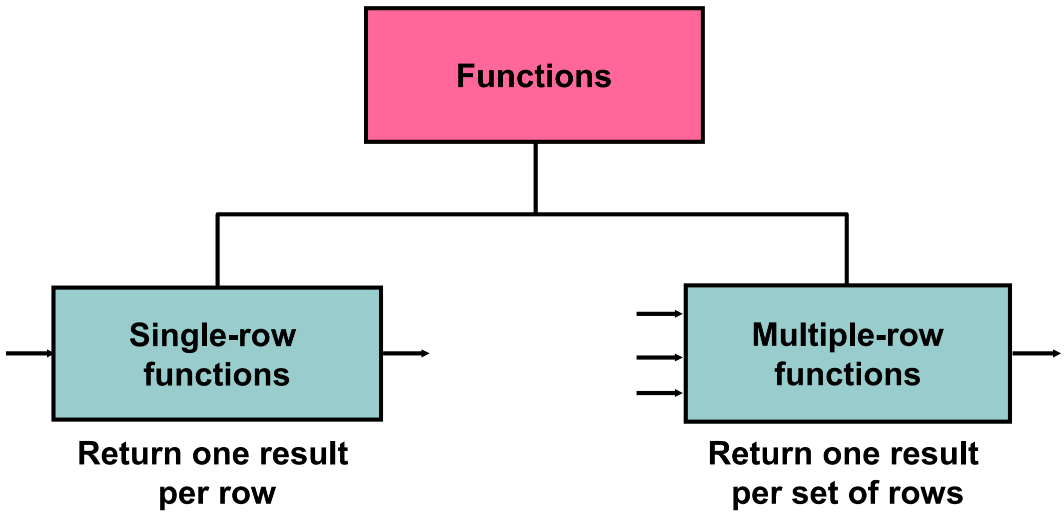
Functions are a very powerful feature of SQL. They can be used to do the following:

- 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.

Note: If you want to know whether a function is a SQL:2003 compliant function, refer to the *Oracle Compliance To Core SQL:2003* section in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Two Types of SQL Functions



Two Types of SQL Functions

There are two types of functions:

- Single-row functions
- Multiple-row functions

Single-Row Functions

These functions operate on single rows only and return one result per row. There are different types of single-row functions. This lesson covers the following ones:

- Character
- Number
- Date
- Conversion
- General

Multiple-Row Functions

Functions can manipulate groups of rows to give one result per group of rows. These functions are also known as *group functions* (covered in lesson 5 titled "Reporting Aggregated Data Using the Group Functions").

Note: For more information and a complete list of available functions and their syntax, see the topic, *Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Single-Row Functions

Single-row functions:

- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

```
function_name [(arg1, arg2,...)]
```

3 - 6

Single-Row Functions

Single-row functions are used to manipulate data items. They accept one or more arguments and return one value for each row that is returned by the query. An argument can be one of the following:

- User-supplied constant
- Variable value
- Column name
- Expression

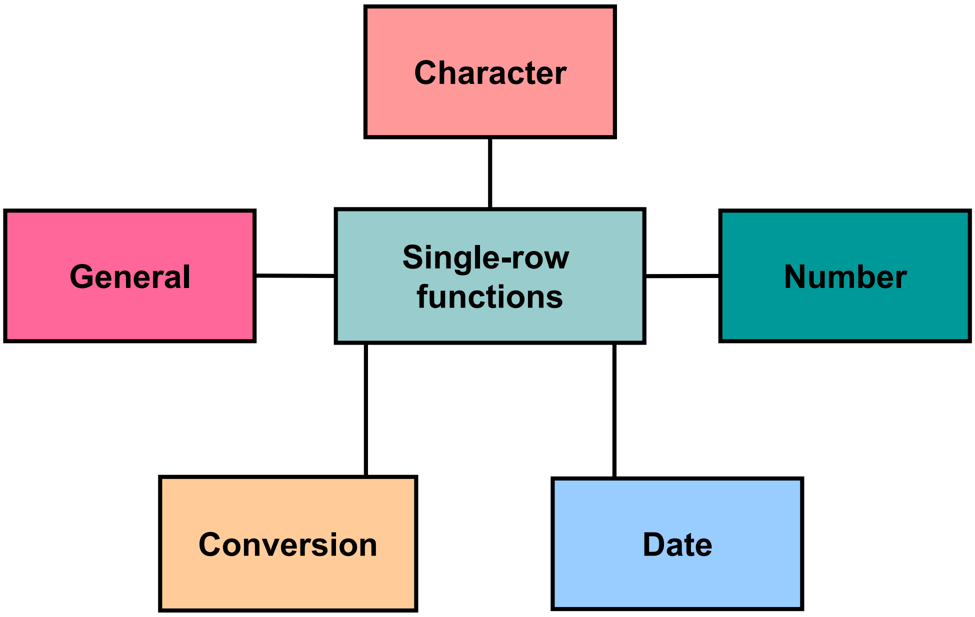
Features of single-row functions include:

- Acting on each row that is returned in the query
- Returning one result per row
- Possibly returning a data value of a different type than the one that is referenced
- Possibly expecting one or more arguments
- Can be used in `SELECT`, `WHERE`, and `ORDER BY` clauses; can be nested

In the syntax:

function_name	is the name of the function
arg1, arg2	is any argument to be used by the function. This can be represented by a column name or expression.

Single-Row Functions



Single-Row Functions (continued)

This lesson covers the following single-row functions:

- **Character functions:** Accept 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 the `DATE` data type except the `MONTHS_BETWEEN` function, which returns a number.)

The following single-row functions are discussed in the next lesson titled “Using Conversion Functions and Conditional Expressions”:

- **Conversion functions:** Convert a value from one data type to another
- **General functions:**
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
  - `CASE`
  - `DECODE`

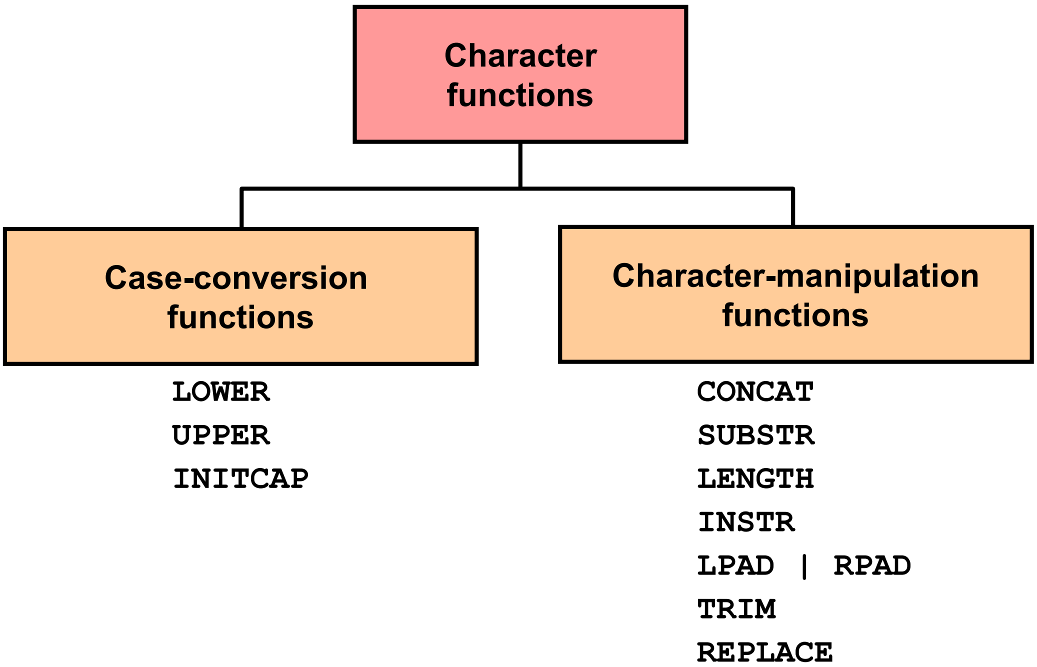
Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date functions

ORACLE

3 - 8

Character Functions



Character Functions

Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following:

- Case-conversion functions
- Character-manipulation functions

Function	Purpose
`LOWER(`*`column`*`|`*`expression`*`)`	Converts alpha character values to lowercase
`UPPER(`*`column`*`|`*`expression`*`)`	Converts alpha character values to uppercase
`INITCAP(`*`column`*`|`*`expression`*`)`	Converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase
`CONCAT(`*`column1`*`|`*`expression1`*`,` *`column2`*`|`*`expression2`*`)`	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.)

Note: The functions discussed in this lesson are only some of the available functions.

Character Functions (continued)

Function	Purpose
`LENGTH(`*column*`|`*expression*`)`	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 string and report the first occurrence.
`LPAD(`*column*`|`*expression*`,` *n*`, '`*string*`')` `RPAD(`*column*`|`*expression*`,` *n*`, '`*string*`')`	Returns an expression left-padded to length of *n* characters with a character expression. Returns an expression right-padded to length of *n* characters with a character expression.
`TRIM(`*leading*`|`*trailing*`|`*both*`,` *trim_character FROM trim_source*`)`	Enables you to trim leading 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 quotation marks. This is a feature that is available in Oracle8*i* and later versions.
`REPLACE(`*text*`,` *search_string*`,` *replacement_string*`)`	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

Note: Some of the functions that are fully or partially SQL:2003 compliant are:

```
UPPER
LOWER
TRIM
LENGTH
SUBSTR
INSTR
```

Refer to the *Oracle Compliance To Core SQL:2003* section in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)* for more information.

Case-Conversion Functions

These functions convert the case for character strings:

Function	Result
`LOWER('SQL Course')`	`sql course`
`UPPER('SQL Course')`	`SQL COURSE`
`INITCAP('SQL Course')`	`Sql Course`

Case-Conversion Functions

`LOWER`, `UPPER`, and `INITCAP` are the three case-conversion functions.

- `LOWER`: Converts mixed-case or uppercase character strings to lowercase
- `UPPER`: Converts mixed-case or lowercase character strings to uppercase
- `INITCAP`: Converts the first letter of each word to uppercase and the remaining letters to lowercase

```
SELECT 'The job id for '||UPPER(last_name)||' is '
 ||LOWER(job_id) AS "EMPLOYEE DETAILS"
FROM   employees;
```

	EMPLOYEE DETAILS
1	The job id for ABEL is sa_rep
2	The job id for DAVIES is st_clerk
3	The job id for DE HAAN is ad_vp
...	
19	The job id for WHALEN is ad_asst
20	The job id for ZLOTKEY is sa_man

Using Case-Conversion Functions

Display the employee number, name, and department number for employee Higgins:

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
```

```
0 rows selected
```

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

Using Case-Conversion Functions

The slide example displays the employee number, name, and department number of employee Higgins.

The `WHERE` clause of the first SQL statement specifies the employee name as `higgins`. Because all the data in the `EMPLOYEES` table is stored in proper case, the name `higgins` does not find a match in the table, and no rows are selected.

The `WHERE` clause of the second SQL statement specifies that the employee name in the `EMPLOYEES` table is compared to `higgins`, converting the `LAST_NAME` column to lowercase for comparison purposes. Because both names are now lowercase, a match is found and one row is selected. The `WHERE` clause can be rewritten in the following manner to produce the same result:

```
...WHERE last_name = 'Higgins'
```

The name in the output appears as it was stored in the database. To display the name in uppercase, use the `UPPER` function in the `SELECT` statement.

```
SELECT employee_id, UPPER(last_name), department_id
FROM   employees
WHERE  INITCAP(last_name) = 'Higgins';
```

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*****`
`REPLACE ('JACK and JUE','J','BL')`	`BLACK and BLUE`
`TRIM('H' FROM 'HelloWorld')`	`elloWorld`

3 - 13

Character-Manipulation Functions

`CONCAT`, `SUBSTR`, `LENGTH`, `INSTR`, `LPAD`, `RPAD`, and `TRIM` are the character-manipulation functions that are covered in this lesson.

- `CONCAT`: Joins values together (You are limited to using two parameters with `CONCAT`.)
- `SUBSTR`: Extracts a string of determined length
- `LENGTH`: Shows the length of a string as a numeric value
- `INSTR`: Finds the numeric position of a named character
- `LPAD`: Returns an expression left-padded to the length of *n* characters with a character expression
- `RPAD`: Returns an expression right-padded to the length of *n* characters with a character expression
- `TRIM`: Trims leading or trailing characters (or both) from a character string (If *`trim_character`* or *`trim_source`* is a character literal, you must enclose it within single quotation marks.)

Note: You can use functions such as `UPPER` and `LOWER` with ampersand substitution. For example, use `UPPER('&job_title')` so that the user does not have to enter the job title in a specific case.

Using the Character-Manipulation Functions

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
       job_id, LENGTH (last_name),
       INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
```

	EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
1	174	EllenAbel	SA_REP	4	0
2	176	JonathonTaylor	SA_REP	6	2
3	178	KimberelyGrant	SA_REP	5	3
4	202	PatFay	MK_REP	3	2

1 2 3

ORACLE

Using the Character-Manipulation Functions

The slide example displays employee first names and last names joined together, the length of the employee last name, and the numeric position of the letter “a” in the employee last name for all employees who have the string, `REP`, contained in the job ID starting at the fourth position of the job ID.

Example:

Modify the SQL statement in the slide to display the data for those employees whose last names end with the letter “n.”

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(last_name, -1, 1) = 'n';
```

	EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
1	102	LexDe Haan	7	5
2	200	JenniferWhalen	6	3
3	201	MichaelHartstein	9	2

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date Functions

ORACLE

Number Functions

- `ROUND`: Rounds value to a specified decimal
- `TRUNC`: Truncates value to a specified decimal
- `MOD`: Returns remainder of division

Function	Result
`ROUND(45.926, 2)`	`45.93`
`TRUNC(45.926, 2)`	`45.92`
`MOD(1600, 300)`	`100`

Number Functions

Number functions accept numeric input and return numeric values. This section describes some of the number functions.

Function	**Purpose**
`ROUND(`*`column`*`|`*`expression`*`, `*`n`*`)`	Rounds the column, expression, or value to *n* decimal places or, if *n* is omitted, no decimal places (If *n* is negative, numbers to the left of decimal point are rounded.)
`TRUNC(`*`column`*`|`*`expression`*`, `*`n`*`)`	Truncates the column, expression, or value to *n* decimal places or, if *n* is omitted, *n* defaults to zero
`MOD(`*`m`*`,`*`n`*`)`	Returns the remainder of *m* divided by *n*

Note: This list contains only some of the available number functions.

For more information, see the section on *Numeric Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1).*

Using the ROUND Function

```
SELECT ROUND(45.923,2), ROUND(45.923,0),
       ROUND(45.923,-1)
FROM   DUAL;
```

	ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
1	45.92	46	50

DUAL is a dummy table that you can use to view results from functions and calculations.

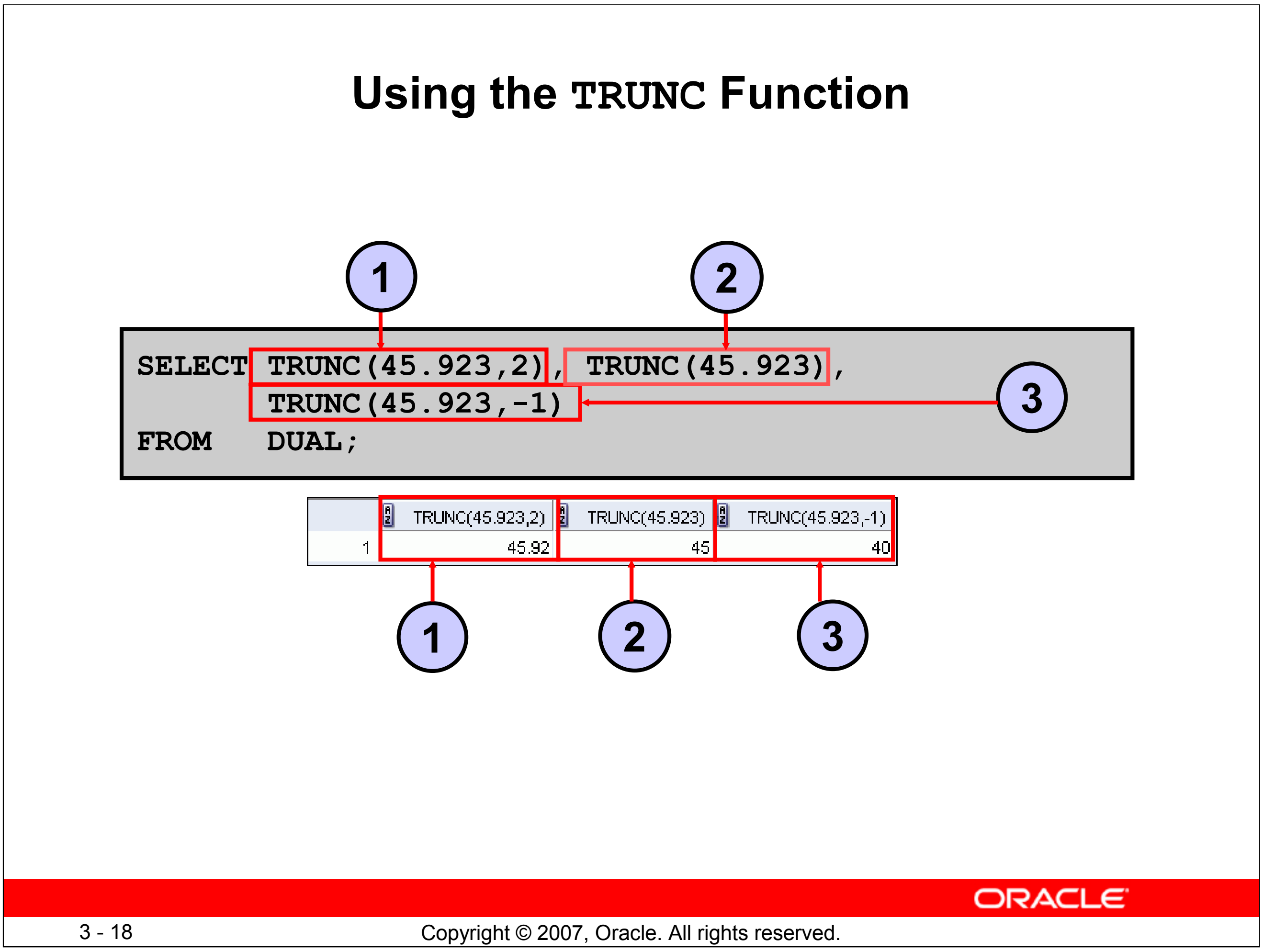
Using the ROUND Function

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 (rounded to the nearest unit of 100).

The ROUND function can also be used with date functions. You will see examples later in this lesson.

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 only once (for example, 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 completeness of the SELECT clause syntax, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from the actual tables.



Using the 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. If the second argument is –1, the value is truncated to one decimal place to the left.

Like the ROUND function, the TRUNC function can be used with date functions.

Using the MOD Function

For all employees with the job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

	LAST_NAME	SALARY	MOD(SALARY,5000)
1	Abel	11000	1000
2	Taylor	8600	3600
3	Grant	7000	2000

Using the MOD Function

The MOD function finds the remainder of the first argument divided by the second argument. The slide example calculates the remainder of the salary after dividing it by 5,000 for all employees whose job ID is SA_REP.

Note: The MOD function is often used to determine whether a value is odd or even.

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date functions

ORACLE

Working with Dates

- The Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.
- The default date display format is DD-MON-RR.
  - Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
  - Enables you to store 20th-century dates in the 21st century in the same way

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-88';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-87
2	Whalen	17-SEP-87

Working with Dates

The Oracle database stores dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds.

The default display and input format for any date is DD-MON-RR. Valid Oracle dates are between January 1, 4712 B.C., and December 31, 9999 A.D.

In the example in the slide, the `HIRE_DATE` column output is displayed in the default format DD-MON-RR. However, dates are not stored in the database in this format. All the components of the date and time are stored. So, although a `HIRE_DATE` such as 17-JUN-87 is displayed as day, month, and year, there is also *time* and *century* information associated with the date. The complete data might be June 17, 1987, 5:10:43 PM.

RR Date Format

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">If the specified two-digit year is:</td></tr>
<tr><td>0–49</td><td>50–99</td></tr>
<tr><td rowspan="2">If two digits of the current year are:</td><td>0–49</td><td>The return date is in the current century</td><td>The return date is in the century before the current one</td></tr>
<tr><td>50–99</td><td>The return date is in the century after the current one</td><td>The return date is in the current century</td></tr>
</table>

RR Date Format

The RR date format is similar to the YY element, but you can use it to specify different centuries. Use the RR date format element instead of YY so that the century of the return value varies according to the specified two-digit year and the last two digits of the current year. The table in 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

Oracle Date Format

This data is stored internally as follows:

CENTURY	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
19	87	06	17	17	10	43

Centuries and the Year 2000

When a record with a date column is inserted into a table, the *century* information is picked up from the `SYSDATE` function. However, when the date column is displayed on the screen, the century component is not displayed (by default).

The `DATE` data type always stores year information as a four-digit number internally: two digits for the century and two digits for the year. For example, the Oracle database stores the year as 1987 or 2004, and not just as 87 or 04.

Using the SYSDATE Function

SYSDATE is a function that returns:

- Date
- Time

```
SELECT sysdate
FROM   dual;
```

	SYSDATE
1	31-MAY-07

Using 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 can display the current date by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL.

Note: SYSDATE returns the current date and time set for the operating system on which the database resides. Hence, if you are in a place in Australia and connected to a remote database in a location in the United States (US), sysdate function will return the US date and time. In that case, you can use the CURRENT_DATE function that returns the current date in the session time zone.

The CURRENT_DATE function and other related time zone functions are discussed in detail in the course titled *Oracle Database 11g: SQL Fundamentals II*.

Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

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.

You can perform the following operations:

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

Using Arithmetic Operators with Dates

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

	LAST_NAME	WEEKS
1	King	1041.168239087301587301587301587302
2	Kochhar	923.02538194444444444444444444444444
3	De Haan	750.168239087301587301587301587302

Using Arithmetic Operators with Dates

The example in the slide displays the last name and the number of weeks employed for all employees in department 90. It subtracts the date on which the employee was hired from the current date (SYSDATE) and divides the result by 7 to calculate the number of weeks that a worker has been employed.

Note: SYSDATE is a SQL function that returns the current date and time. Your results may differ depending on the date and time set for the operating system of your local database when you run the SQL query.

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

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date functions

ORACLE

Date-Manipulation Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

Date-Manipulation Functions

Date functions operate on Oracle dates. All date functions return a value of the 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*

The above list is a subset of the available date functions. ROUND and TRUNC number functions can also be used to manipulate the date values as shown below:

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

The format models are covered in detail in the next lesson titled “Using Conversion Functions and Conditional Expressions.”

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('31-JAN-96',1)	'29-FEB-96'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

Using Date Functions

In the slide example, the `ADD_MONTHS` function adds one month to the supplied date value, "31-JAN-96" and returns "29-FEB-96." The function recognizes the year 1996 as the leap year and hence returns the last day of the February month. If you change the input date value to "31-JAN-95," the function returns "28-FEB-95."

For example, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and the last day of the hire month for all employees who have been employed for fewer than 100 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) < 100;
```

	EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(HIRE_DATE,'FRIDAY')	LAST_DAY(HIRE_DATE)
1	124	16-NOV-99	91.1099600...	16-MAY-00	19-NOV-99	30-NOV-99
2	149	29-JAN-00	88.6906052...	29-JUL-00	04-FEB-00	31-JAN-00
3	178	24-MAY-99	96.8518955...	24-NOV-99	28-MAY-99	31-MAY-99
4	99999	07-JUN-99	96.4002826...	07-DEC-99	11-JUN-99	30-JUN-99
5	113	11-JUN-07	0.25824335...	11-DEC-07	15-JUN-07	30-JUN-07

Using ROUND and TRUNC Functions with Dates

Assume SYSDATE = '25-JUL-03':

Function	Result
ROUND(SYSDATE,'MONTH')	01-AUG-03
ROUND(SYSDATE ,'YEAR')	01-JAN-04
TRUNC(SYSDATE ,'MONTH')	01-JUL-03
TRUNC(SYSDATE ,'YEAR')	01-JAN-03

ORACLE

Using ROUND and TRUNC Functions with Dates

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. If the format model is month, dates 1-15 result in the first day of the current month. Dates 16-31 result in the first day of the next month. If the format model is year, months 1-6 result in January 1 of the current year. Months 7-12 result in January 1 of the next year.

Example:

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and starting month 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';
```

	EMPLOYEE_ID	HIRE_DATE	ROUND(HIRE_DATE,'MONTH')	TRUNC(HIRE_DATE,'MONTH')
1	142	29-JAN-97	01-FEB-97	01-JAN-97
2	202	17-AUG-97	01-SEP-97	01-AUG-97

Summary

In this lesson, you should have learned how to:

- Perform calculations on data using functions
- Modify individual data items using functions

3 - 31

Summary

Single-row functions can be nested to any level. Single-row functions can manipulate the following:

- Character data: `LOWER`, `UPPER`, `INITCAP`, `CONCAT`, `SUBSTR`, `INSTR`, `LENGTH`
- Number data: `ROUND`, `TRUNC`, `MOD`
- Date values: `SYSDATE`, `MONTHS_BETWEEN`, `ADD_MONTHS`, `NEXT_DAY`, `LAST_DAY`

Remember the following:

- Date values can also use arithmetic operators.
- `ROUND` and `TRUNC` functions can also be used with date values.

SYSDATE and DUAL

`SYSDATE` is a date function that returns the current date and time. It is customary to select `SYSDATE` from a dummy table called `DUAL`.

Practice 3: Overview

This practice covers the following topics:

- Writing a query that displays the current date
- Creating queries that require the use of numeric, character, and date functions
- Performing calculations of years and months of service for an employee

3 - 32

Practice 3: Overview

This practice provides a variety of exercises using different functions that are available for character, number, and date data types.

Practice 3

Part 1

1. Write a query to display the system date. Label the column as `Date`.

   Note: If your database is remotely located in a different time zone, the output will be the date for the operating system on which the database resides.

	Date
1	31-MAY-07

2. The HR department needs a report to display the employee number, last name, salary, and salary increased by 15.5% (expressed as a whole number) for each employee. Label the column `New Salary`. Save your SQL statement in a file named `lab_03_02.sql`.

3. Run your query in the `lab_03_02.sql` file.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary
1	100	King	24000	27720
2	101	Kochhar	17000	19635
3	102	De Haan	17000	19635
4	103	Hunold	9000	10395
5	104	Ernst	6000	6930
6	107	Lorentz	4200	4851
7	124	Mourgos	5800	6699
8	141	Rajs	3500	4043
9	142	Davies	3100	3581
10	143	Matos	2600	3003
...				
19	205	Higgins	12000	13860
20	206	Gietz	8300	9587

4. Modify your query `lab_03_02.sql` to add a column that subtracts the old salary from the new salary. Label the column `Increase`. Save the contents of the file as `lab_03_04.sql`. Run the revised query.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary	Increase
1	100	King	24000	27720	3720
2	101	Kochhar	17000	19635	2635
3	102	De Haan	17000	19635	2635
4	103	Hunold	9000	10395	1395
5	104	Ernst	6000	6930	930
...					
20	206	Gietz	8300	9587	1287

Practice 3 (continued)

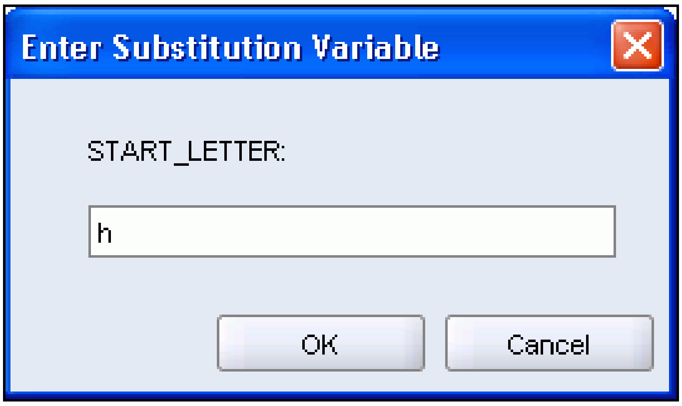
5. Write a query that displays the last name (with the first letter in uppercase and all the other letters in lowercase) and the length of the last name for all employees whose name starts with the letters "J," "A," or "M." Give each column an appropriate label. Sort the results by the employees' last names.

	Name	Length
1	Abel	4
2	Matos	5
3	Mourgos	7

Rewrite the query so that the user is prompted to enter a letter that the last name starts with. For example, if the user enters "`H`" (capitalized) when prompted for a letter, then the output should show all employees whose last name starts with the letter "H."

	Name	Length
1	Hartstein	9
2	Higgins	7
3	Hunold	6

Modify the query such that the case of the entered letter does not affect the output. The entered letter must be capitalized before being processed by the `SELECT` query.



	Name	Length
1	Hartstein	9
2	Higgins	7
3	Hunold	6

Practice 3 (continued)

6. The HR department wants to find the duration of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column as `MONTHS_WORKED`. Order your results by the number of months employed. Round the number of months up to the closest whole number.
   Note: Because this query depends on the date when it was executed, the values in the `MONTHS_WORKED` column will differ for you.

	LAST_NAME	MONTHS_WORKED
1	Zlotkey	88
2	Mourgos	90
3	Grant	96
4	Lorentz	100
5	Vargas	107
6	Taylor	110
7	Matos	111
8	Fay	117
9	Davies	124
10	Abel	133
11	Hartstein	135
12	Rajs	139
13	Higgins	156
14	Gietz	156
15	De Haan	173
16	Ernst	192
17	Hunold	209
18	Kochhar	212
19	Whalen	236
20	King	239

Practice 3 (continued)

If you have time, complete the following exercises:

7. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the $ symbol. Label the column as `SALARY`.

	LAST_NAME	SALARY
1	King	$$$$$$$$$$24000
2	Kochhar	$$$$$$$$$$17000

...

20	Gietz	$$$$$$$$$$$8300

8. Create a query that displays the first eight characters of the employees' last names and indicates the amounts of their salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column as `EMPLOYEES_AND_THEIR_SALARIES`.

	EMPLOYEES_AND_THEIR_SALARIES
1	King ************************
2	Kochhar *****************
3	De Haan *****************
4	Hartstei **************
5	Higgins ************

...

19	Matos **
20	Vargas **

9. Create a query to display the last name and the number of weeks employed for all employees in department 90. Label the number of weeks column as `TENURE`. Truncate the number of weeks value to 0 decimal places. Show the records in descending order of the employee's tenure.
Note: The `TENURE` value will differ as it depends on the date on which you run the query.

	LAST_NAME	TENURE
1	King	1041
2	Kochhar	923
3	De Haan	750

Using Conversion Functions and Conditional Expressions

Objectives

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

- Describe various types of conversion functions that are available in SQL
- Use the `TO_CHAR`, `TO_NUMBER`, and `TO_DATE` conversion functions
- Apply conditional expressions in a `SELECT` statement

4 - 2

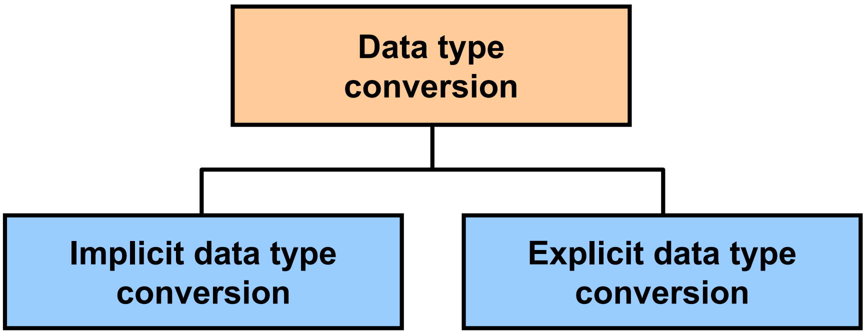
Objectives

This lesson focuses on functions that convert data from one type to another (for example, conversion from character data to numeric data) and discusses the conditional expressions in SQL `SELECT` statements.

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- Nesting functions
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - `DECODE`

Conversion Functions



4 - 4

Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle database can be defined by using the American National Standards Institute (ANSI), DB2, and SQL/DS data types. However, the Oracle server internally converts such data types to Oracle data types.

In some cases, the Oracle server receives data of one data type where it expects data of a different data type. When this happens, 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. Conversion functions convert a value from one data type to another. Generally, the form of the function names follows the convention `data type TO data type`. The first data type is the input data type and the second data type is the output.

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

Implicit Data Type Conversion

In expressions, the Oracle server can automatically convert the following:

From	To
`VARCHAR2 or CHAR`	`NUMBER`
`VARCHAR2 or CHAR`	`DATE`

Implicit Data Type Conversion

Oracle server can automatically perform data type conversion in an expression. For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string `'01-JAN-90'` to a date. Therefore, a `VARCHAR2` or `CHAR` value can be implicitly converted to a number or date data type in an expression.

Implicit Data Type Conversion

For expression evaluation, the Oracle server can automatically convert the following:

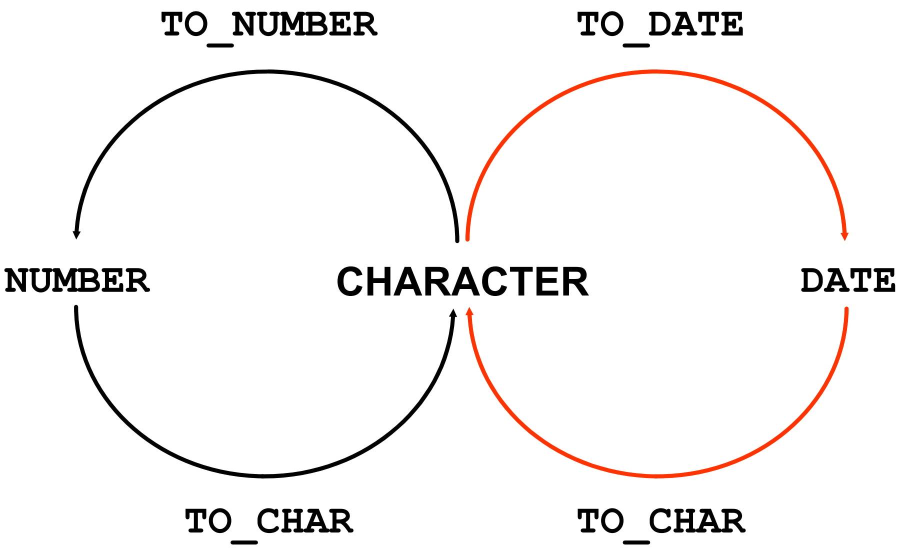
From	To
NUMBER	VARCHAR2 or CHAR
DATE	VARCHAR2 or CHAR

Implicit Data Type Conversion (continued)

In general, the Oracle server uses the rule for expressions when a data type conversion is needed. For example, the expression `grade = 2` results in the implicit conversion of the number `20000` to the string "2" because grade is a `CHAR(2)` column.

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

Explicit Data Type Conversion

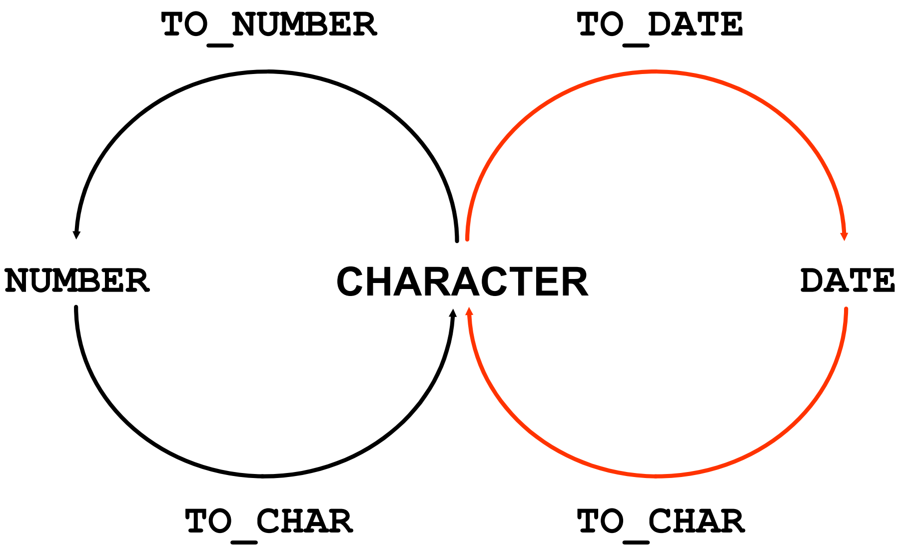


Explicit Data Type Conversion

SQL provides three functions to convert a value from one data type to another:

Function	Purpose
`TO_CHAR(`*`number`*`|`*`date`*`,[ `*`fmt`*`], [`*`nlsparams`*`])`	Converts a number or date value to a `VARCHAR2` character string with the format model *`fmt`* **Number conversion:** The `nlsparams` parameter specifies the following characters, which are returned by number format elements: • Decimal character • Group separator • Local currency symbol • International currency symbol If `nlsparams` or any other parameter is omitted, this function uses the default parameter values for the session.

Explicit Data Type Conversion



Explicit Data Type Conversion (continued)

Function	Purpose
`TO_CHAR(`*number*`|`*date*`,[ `*fmt*`], [`*nlsparams*`])`	**Date conversion:** The `nlsparams` parameter specifies the language in which the 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*. The `nlsparams` parameter has the same purpose in this function as in the `TO_CHAR` function for number conversion.
`TO_DATE(`*char*`,[`*fmt*`],[`*nlsparams*`])`	Converts a character string representing a date to a date value according to the *fmt* that is specified. If *fmt* is omitted, the format is DD-MON-YY. The `nlsparams` parameter has the same purpose in this function as in the `TO_CHAR` function for date conversion.

Explicit Data Type Conversion (continued)

Note: The list of functions mentioned in this lesson includes only some of the available conversion functions.

For more information, see the section on *Conversion Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- Nesting functions
- General functions:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- Conditional expressions:
  - CASE
  - DECODE

ORACLE

4 - 10

Using the `TO_CHAR` Function with Dates

```
TO_CHAR(date, 'format_model')
```

The format model:

- Must be enclosed with single quotation marks
- Is case-sensitive
- Can include any valid date format element
- Has an `fm` element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

Using the `TO_CHAR` Function with Dates

`TO_CHAR` converts a datetime data type to a value of `VARCHAR2` data type in the format specified by the *format_model*. A format model is a character literal that describes the format of datetime stored in a character string. For example, the datetime format model for the string `'11-Nov-1999'` is `'DD-Mon-YYYY'`. You can use the `TO_CHAR` function to convert a date from its default format to the one that you specify.

Guidelines

- The format model must be enclosed with single quotation marks and is case-sensitive.
- The format model can include any valid date format element. But be sure to separate the date value from the format model with 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';
```

	EMPLOYEE_ID	MONTH_HIRED
1	205	06/94

Elements of the Date Format Model

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for the month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the 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	Description
SCC or CC	Century; server prefixes B.C. date with -
Years in dates YYYY or SYYYY	Year; server prefixes B.C. date with -
YYY or YY or Y	Last three, two, or one digit of the year
Y,YYY	Year with comma in this position
IYYY, IYY, IY, I	Four-, three-, two-, or one-digit year based on the ISO standard
SYEAR or YEAR	Year spelled out; server prefixes B.C. date with -
BC or AD	Indicates B.C. or A.D. year
B.C. or A.D.	Indicates B.C. or A.D. year using periods
Q	Quarter of year
MM	Month: two-digit value
MONTH	Name of the month padded with blanks to a length of nine characters
MON	Name of the month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of the year or month
DDD or DD or D	Day of the year, month, or week
DAY	Name of the day padded with blanks to a length of nine characters
DY	Name of the day; three-letter abbreviation
J	Julian day; the number of days since December 31, 4713 B.C.
IW	Weeks in the year from ISO standard (1 to 53)

Elements of the Date Format Model

- Time elements format the time portion of the date:

`HH24:MI:SS AM`	`15:45:32 PM`

- Add character strings by enclosing them with double quotation marks:

`DD "of" MONTH`	`12 of OCTOBER`

- Number suffixes spell out numbers:

`ddspth`	`fourteenth`

Elements of the Date Format Model

Use the formats that are listed in the following tables to display time information and literals, and to change numerals to spelled numbers.

Element	Description
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12 or HH24	Hour of day, or hour (1–12), or hour (0–23)
MI	Minute (0–59)
SS	Second (0–59)
SSSSS	Seconds past midnight (0–86399)

Other Formats

Element	Description
/ . ,	Punctuation is reproduced in the result.
“of the”	Quoted string is reproduced in the result.

Specifying Suffixes to Influence Number Display

Element	Description
TH	Ordinal number (for example, DDTH for 4TH)
SP	Spelled-out number (for example, DDSP for FOUR)
SPTH or THSP	Spelled-out ordinal numbers (for example, DDSPTH for FOURTH)

Using the `TO_CHAR` Function with Dates

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

	LAST_NAME	HIREDATE
1	King	17 June 1987
2	Kochhar	21 September 1989
3	De Haan	13 January 1993
4	Hunold	3 January 1990
5	Ernst	21 May 1991
6	Lorentz	7 February 1999
7	Mourgos	16 November 1999
8	Rajs	17 October 1995
9	Davies	29 January 1997
10	Matos	15 March 1998
...		
19	Higgins	7 June 1994
20	Gietz	7 June 1994

4 - 16

Using the `TO_CHAR` Function with Dates

The SQL statement in the slide displays the last names and hire dates for all the employees. The hire date appears as 17 June 1987.

Example:

Modify the example in the slide to display the dates in a format that appears as “Seventeenth of June 1987 12:00:00 AM.”

```
SELECT  last_name,
 TO_CHAR(hire_date,
      'fmDdspth "of" Month YYYY fmHH:MI:SS AM')
 HIREDATE
FROM    employees;
```

	LAST_NAME	HIREDATE
1	King	Seventeenth of June 1987 12:00:00 AM
2	Kochhar	Twenty-First of September 1989 12:00:00 AM

...

Notice that the month follows the format model specified; in other words, the first letter is capitalized and the rest are in lowercase.

Using the TO_CHAR Function with Numbers

```
TO_CHAR(number, 'format_model')
```

These are some of the format elements that you can use with the TO_CHAR function to display a number value as a character:

Element	Result
9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as a thousands indicator

ORACLE

4 - 17

Using the TO_CHAR Function with Numbers

When working with number values, such as character strings, you should 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.

Using the `TO_CHAR` Function with Numbers (continued)

Number Format Elements

If you are converting a number to the character data type, you can use the following format elements:

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
$	Floating dollar sign	$999999	$1234
L	Floating local currency symbol	L999999	FF1234
D	Returns the decimal character in the specified position. The default is a period (.).	99D99	99.99
.	Decimal point in position specified	999999.99	1234.00
G	Returns the group separator in the specified position. You can specify multiple group separators in a number format model.	9,999	9G999
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
U	Returns in the specified position the “Euro” (or other) dual currency	U9999	€1234
V	Multiply by 10 *n* times (*n* = number of 9s after V)	9999V99	123400
S	Returns the negative or positive value	S9999	-1234 or +1234
B	Display zero values as blank, not 0	B9999.99	1234.00

Using the TO_CHAR Function with Numbers

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

	SALARY
1	$6,000.00

Using the TO_CHAR Function with Numbers (continued)

- The Oracle server displays a string of number signs (#) in place of a whole number whose digits exceed the number of digits provided in the format model.
- The Oracle server rounds the stored decimal value to the number of decimal places provided in the format model.

Using the TO_NUMBER and TO_DATE Functions

- Convert a character string to a number format using the `TO_NUMBER` function:

```
TO_NUMBER(char[, 'format_model'])
```

- Convert a character string to a date format using the `TO_DATE` function:

```
TO_DATE(char[, 'format_model'])
```

- These functions have an `fx` modifier. This modifier specifies the exact match for the character argument and date format model of a `TO_DATE` function.

4 - 20

Using 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, use the `TO_NUMBER` or `TO_DATE` functions. The format model that you select is based on the previously demonstrated format elements.

The `fx` modifier specifies the exact match 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`, the numbers in the character argument can omit leading zeros.

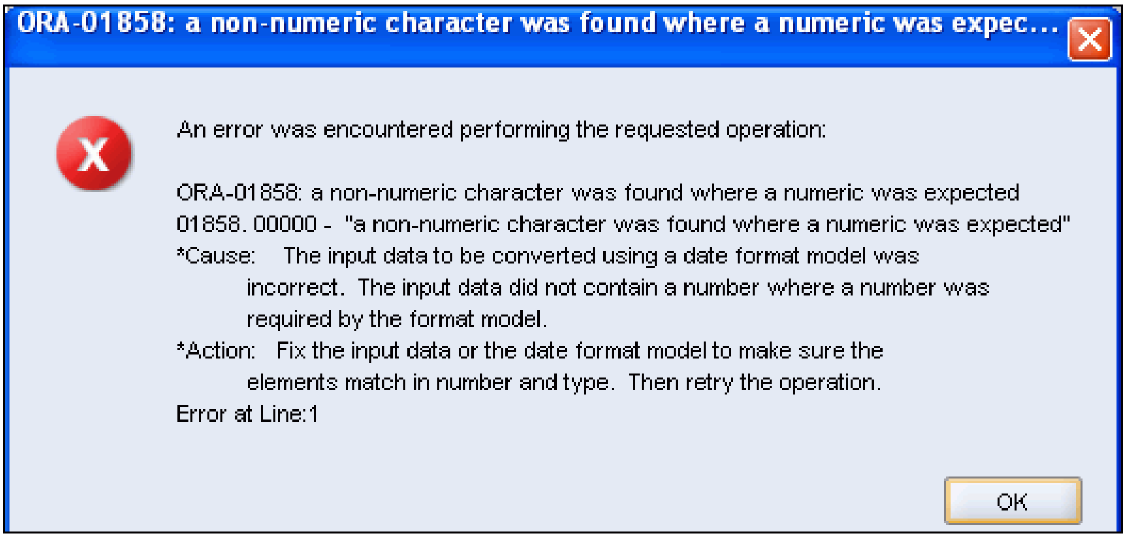
Using the `TO_NUMBER` and `TO_DATE` Functions (continued)

Example:

Display the name and hire date for all employees who started on May 24, 1999. There are two spaces after the month *May* and the number *24* in the following example. Because the `fx` modifier is used, an exact match is required and the spaces after 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 error:



Using the TO_CHAR and TO_DATE Function with RR Date Format

To find employees hired before 1990, use the RR date format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM  employees
WHERE hire_date < TO_DATE('01-Jan-90','DD-Mon-RR');
```

	LAST_NAME	TO_CHAR(HIRE_DATE,'DD-MON-YYYY')
1	King	17-Jun-1987
2	Kochhar	21-Sep-1989
3	Whalen	17-Sep-1987

Using the TO_CHAR and TO_DATE Function with RR Date Format

To find employees who were hired before 1990, the RR format can be used. Because the current year is greater than 1999, the RR format interprets the year portion of the date from 1950 to 1999.

The following command, on the other hand, results in no rows being selected because the YY format interprets the year portion of the date in the current century (2090).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM   employees
WHERE  TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
```

```
0 rows selected
```

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- Nesting functions
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - `DECODE`

ORACLE

Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from the deepest level to the least deep level.

```
F3(F2(F1(col,arg1),arg2),arg3)
```

Step 1 = Result 1

Step 2 = Result 2

Step 3 = Result 3

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.

Nesting Functions

```
SELECT last_name,
  UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
FROM   employees
WHERE  department_id = 60;
```

	LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

Nesting Functions (continued)

The slide example displays the last names of employees in department 60. The evaluation of the SQL statement involves three steps:

1. The inner function retrieves the first eight characters of the last name.
   `Result1 = SUBSTR (LAST_NAME, 1, 8)`
2. The outer function concatenates the result with `_US`.
   `Result2 = CONCAT(Result1, '_US')`
3. The outermost function converts the results to uppercase.

The entire expression becomes the column heading because no column alias was given.

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;
```

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- Nesting functions
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - `DECODE`

General Functions

The following functions work with any data type and pertain to using nulls:

- `NVL (expr1, expr2)`
- `NVL2 (expr1, expr2, expr3)`
- `NULLIF (expr1, expr2)`
- `COALESCE (expr1, expr2, ..., exprn)`

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 expr1 can have any data type.
NULLIF	Compares two expressions and returns null if they are equal; returns the first expression if they are not equal
COALESCE	Returns the first non-null expression in the expression list

Note: For more information about the hundreds of functions available, see the section on *Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

NVL Function

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match:
  - `NVL(commission_pct,0)`
  - `NVL(hire_date,'01-JAN-97')`
  - `NVL(job_id,'No Job Yet')`

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*.

NVL Conversions for Various Data Types

Data Type	Conversion Example
NUMBER	NVL(*number_column*,9)
DATE	NVL(*date_column*, '01-JAN-95')
CHAR or VARCHAR2	NVL(*character_column*, 'Unavailable')

Using the NVL Function

```
SELECT last_name, salary, NVL(commission_pct, 0),
   (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200
11	Vargas	2500	0	30000
12	Zlotkey	10500	0.2	151200

...

1 2

4 - 29

Using the NVL Function

To calculate the annual compensation of all employees, you need to multiply the monthly salary by 12 and then add the commission percentage to the result:

```
SELECT last_name, salary, commission_pct,
 (salary*12) + (salary*12*commission_pct) AN_SAL
FROM   employees;
```

	LAST_NAME	SALARY	COMMISSION_PCT	AN_SAL
1	King	24000	(null)	(null)
...				
11	Vargas	2500	(null)	(null)
12	Zlotkey	10500	0.2	151200
13	Abel	11000	0.3	171600

...

Notice that the annual compensation is calculated for only those employees who earn a commission. If any column value in an expression is null, the result is null. To calculate values for all employees, you must convert the null value to a number before applying the arithmetic operator. In the example in the slide, the NVL function is used to convert null values to zero.

Using the NVL2 Function

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

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM

Using 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

```
NVL2(expr1, expr2, expr3)
```

In the syntax:

- *expr1* is the source value or expression that may contain a null
- *expr2* is the value that is returned if *expr1* is not null
- *expr3* is the value that is returned if *expr1* is null

In the example shown in the slide, 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 value, 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 the latter 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.

Using the NULLIF Function

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

	FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
1	Ellen	5	Abel	4	5
2	Curtis	6	Davies	6	(null)
3	Lex	3	De Haan	7	3
4	Bruce	5	Ernst	5	(null)
5	Pat	3	Fay	3	(null)
6	William	7	Gietz	5	7
7	Kimberely	9	Grant	5	9
...					
19	Jennifer	8	Whalen	6	8
20	Eleni	5	Zlotkey	7	5

1 2 3

ORACLE

Using the NULLIF Function

The NULLIF function compares two expressions. If they are equal, the function returns a null. If they are not equal, the function returns the first expression. However, you cannot specify the literal NULL for the first expression.

Syntax

```
NULLIF (expr1, expr2)
```

In the syntax:

- NULLIF compares *expr1* and *expr2*. If they are equal, then the function returns null. If they are not, then the function returns *expr1*. However, you cannot specify the literal NULL for *expr1*.

In the example shown in the slide, the length of the first name in the EMPLOYEES table is compared to the length of the last name in the EMPLOYEES table. When the lengths of the names are equal, a null value is displayed. When the lengths of the names are not equal, the length of the first name is displayed.

Note: The NULLIF function is logically equivalent to the following CASE expression. The CASE expression is discussed on a subsequent page:

```
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END
```

Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternate values.
- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.

Using the COALESCE Function

The COALESCE function returns the first non-null 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

Note that all expressions must be of the same data type.

Using the COALESCE Function

```
SELECT last_name, employee_id,
COALESCE(TO_CHAR(commission_pct),TO_CHAR(manager_id),
         'No commission and no manager')
FROM employees;
```

	LAST_NAME	EMPLOYEE_ID	COALESCE(TO_CHAR(COM
1	King	100	No commission and no manager
2	Kochhar	101	100
3	De Haan	102	100
4	Hunold	103	102
5	Ernst	104	103
6	Lorentz	107	103
7	Mourgos	124	100
8	Rajs	141	124

...

12	Zlotkey	149	.2
13	Abel	174	.3
14	Taylor	176	.2
15	Grant	178	.15
16	Whalen	200	101

...

Using the COALESCE Function (continued)

In the example shown in the slide, if the `manager_id` value is not null, it is displayed. If the `manager_id` value is null, then the `commission_pct` is displayed. If the `manager_id` and `commission_pct` values are null, then “No commission and no manager” is displayed. Note, `TO_CHAR` function is applied so that all expressions are of the same data type.

Using the `COALESCE` Function (continued)

Example:

For the employees who do not get any commission, your organization wants to give a salary increment of $2,000 and for employees who get commission, the query should compute the new salary that is equal to the existing salary added to the commission amount.

```
SELECT last_name, salary, commission_pct,
 COALESCE((salary+(commission_pct*salary)), salary+2000, salary) "New
  Salary"
FROM   employees;
```

Note: Examine the output. For employees who do not get any commission, the New Salary column shows the salary incremented by $2,000 and for employees who get commission, the New Salary column shows the computed commission amount added to the salary.

	LAST_NAME	SALARY	COMMISSION_PCT	New Salary
1	King	24000	(null)	26000
2	Kochhar	17000	(null)	19000
3	De Haan	17000	(null)	19000
4	Hunold	9000	(null)	11000

...

9	Davies	3100	(null)	5100
10	Matos	2600	(null)	4600
11	Vargas	2500	(null)	4500
12	Zlotkey	10500	0.2	12600
13	Abel	11000	0.3	14300
14	Taylor	8600	0.2	10320
15	Grant	7000	0.15	8050
16	Whalen	4400	(null)	6400
17	Hartstein	13000	(null)	15000
18	Fay	6000	(null)	8000
19	Higgins	12000	(null)	14000
20	Gietz	8300	(null)	10300

...

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- Nesting functions
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - `DECODE`

ORACLE

Conditional Expressions

- Provide the use of the `IF-THEN-ELSE` logic within a SQL statement
- Use two methods:
  - `CASE` expression
  - `DECODE` function

ORACLE

4 - 36

Conditional Expressions

The two methods that are used to implement conditional processing (`IF-THEN-ELSE` logic) in a SQL statement are the `CASE` expression and the `DECODE` function.

Note: The `CASE` expression complies with the ANSI SQL. The `DECODE` function is specific to Oracle syntax.

CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
         [WHEN comparison_expr2 THEN return_expr2
          WHEN comparison_exprn THEN return_exprn
          ELSE else_expr]
END
```

ORACLE

CASE Expression

CASE expressions allow you to use the 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 if an ELSE clause exists, then the Oracle server returns else_expr. Otherwise, the Oracle server returns a 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.

Using the `CASE` Expression

Facilitates conditional inquiries by doing the work of an `IF-THEN-ELSE` statement:

```
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;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
...				

Using the `CASE` Expression

In the SQL statement in the slide, 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.

This is an example of a searched `CASE` expression. In a searched `CASE` expression, the search occurs from left to right until an occurrence of the listed condition is found, and then it returns the return expression. If no condition is found to be true, and if an `ELSE` clause exists, the return expression in the `ELSE` clause is returned; otherwise, a `NULL` is returned.

```
SELECT last_name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

DECODE Function

Facilitates conditional inquiries by doing the work of a CASE expression or an IF-THEN-ELSE statement:

```
DECODE(col|expression, search1, result1
                       [, search2, result2,...,]
                       [, default])
```


DECODE Function

The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic that is 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.

Using the DECODE Function

```
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;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
...				

ORACLE

4 - 40

Using the DECODE Function

In the SQL statement in the slide, the value of JOB_ID is tested. 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 expressed in pseudocode as an IF-THEN-ELSE statement:

```
IF job_id = 'IT_PROG'     THEN   salary = salary*1.10
IF job_id = 'ST_CLERK'    THEN   salary = salary*1.15
IF job_id = 'SA_REP'      THEN   salary = salary*1.20
ELSE salary = salary
```

Using the DECODE Function

Display the applicable tax rate for each employee in department 80:

```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
                         0, 0.00,
                         1, 0.09,
                         2, 0.20,
                         3, 0.30,
                         4, 0.40,
                         5, 0.42,
                         6, 0.44,
                            0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```


Using the DECODE Function (continued)

This slide shows another example using the DECODE function. In this example, you determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as follows:

Monthly Salary Range	*Tax Rate*
$0.00–1,999.99	00%
$2,000.00–3,999.99	09%
$4,000.00–5,999.99	20%
$6,000.00–7,999.99	30%
$8,000.00–9,999.99	40%
$10,000.00–11,999.99	42%
$12,200.00–13,999.99	44%
$14,000.00 or greater	45%

	LAST_NAME	SALARY	TAX_RATE
1	Zlotkey	10500	0.42
2	Abel	11000	0.42
3	Taylor	8600	0.4

Summary

In this lesson, you should have learned how to:
- Alter date formats for display using functions
- Convert column data types using functions
- Use `NVL` functions
- Use `IF-THEN-ELSE` logic and other conditional expressions in a `SELECT` statement

Summary

Remember the following:
- Conversion functions can convert character, date, and numeric values: `TO_CHAR`, `TO_DATE`, `TO_NUMBER`
- There are several functions that pertain to nulls, including `NVL`, `NVL2`, `NULLIF`, and `COALESCE`.
- `IF-THEN-ELSE` logic can be applied within a SQL statement by using the `CASE` expression or the `DECODE` function.

Practice 4: Overview

This practice covers the following topics:

- Creating queries that use TO_CHAR, TO_DATE, and other DATE functions
- Creating queries that use conditional expressions such as DECODE and CASE

Practice 4: Overview

This practice provides a variety of exercises using TO_CHAR and TO_DATE functions, and conditional expressions such as DECODE and CASE. Remember that for nested functions, the results are evaluated from the innermost function to the outermost function.

Practice 4

1. Create a report that produces the following for each employee:
   `<employee last name> earns <salary> monthly but wants <3 times salary.>`. Label the column `Dream Salaries`.

	Dream Salaries
1	King earns $24,000.00 monthly but wants $72,000.00.
2	Kochhar earns $17,000.00 monthly but wants $51,000.00.
3	De Haan earns $17,000.00 monthly but wants $51,000.00.
4	Hunold earns $9,000.00 monthly but wants $27,000.00.
5	Ernst earns $6,000.00 monthly but wants $18,000.00.

...

19	Higgins earns $12,000.00 monthly but wants $36,000.00.
20	Gietz earns $8,300.00 monthly but wants $24,900.00.

2. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column `REVIEW`. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

	LAST_NAME	HIRE_DATE	REVIEW
1	King	17-JUN-87	Monday, the Twenty-First of December, 1987
2	Kochhar	21-SEP-89	Monday, the Twenty-Sixth of March, 1990
3	De Haan	13-JAN-93	Monday, the Nineteenth of July, 1993
4	Hunold	03-JAN-90	Monday, the Ninth of July, 1990
5	Ernst	21-MAY-91	Monday, the Twenty-Fifth of November, 1991

...

19	Higgins	07-JUN-94	Monday, the Twelfth of December, 1994
20	Gietz	07-JUN-94	Monday, the Twelfth of December, 1994

Practice 4 (continued)

3. Display the last name, hire date, and day of the week on which the employee started. Label the column `DAY`. Order the results by the day of the week, starting with Monday.

	LAST_NAME	HIRE_DATE	DAY
1	Grant	24-MAY-99	MONDAY
2	Gietz	07-JUN-94	TUESDAY
3	Taylor	24-MAR-98	TUESDAY
4	Higgins	07-JUN-94	TUESDAY
5	Rajs	17-OCT-95	TUESDAY

...

19	Lorentz	07-FEB-99	SUNDAY
20	Fay	17-AUG-97	SUNDAY

4. Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, show "No Commission." Label the column `COMM`.

	LAST_NAME	COMM
1	King	No Commission
2	Kochhar	No Commission
3	De Haan	No Commission
4	Hunold	No Commission
5	Ernst	No Commission
6	Lorentz	No Commission

...

12	Zlotkey	.2
13	Abel	.3
14	Taylor	.2
15	Grant	.15
16	Whalen	No Commission
17	Hartstein	No Commission
18	Fay	No Commission
19	Higgins	No Commission
20	Gietz	No Commission

Practice 4 (continued)

If you have time, complete the following exercises:

5. Using the `DECODE` function, write a query that displays the grade of all employees based on the value of the column `JOB_ID`, using the following data:

Job	***Grade***
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
None of the above	0

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AC_MGR	0
3	AD_ASST	0
4	AD_PRES	A
5	AD_VP	0

…

18	ST_CLERK	E
19	ST_CLERK	E
20	ST_MAN	B

6. Rewrite the statement in the preceding exercise using the `CASE` syntax.

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AC_MGR	0
3	AD_ASST	0
4	AD_PRES	A
5	AD_VP	0

…

18	ST_CLERK	E
19	ST_CLERK	E
20	ST_MAN	B

5

Reporting Aggregated Data Using the Group Functions

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 by using the `GROUP BY` clause
- Include or exclude grouped rows by using the `HAVING` clause

ORACLE

5 - 2

Objectives

This lesson further addresses functions. It focuses on obtaining summary information (such as averages) for groups of rows. It discusses how to group rows in a table into smaller sets and how to specify search criteria for groups of rows.

Lesson Agenda

- Group functions:
  – Types and syntax
  – Use `AVG`, `SUM`, `MIN`, `MAX`, `COUNT`
  – Use `DISTINCT` keyword within group functions
  – `NULL` values in a group function
- Grouping rows:
  – `GROUP BY` clause
  – `HAVING` clause
- Nesting group functions

What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600
...		
18	20	6000
19	110	12000
20	110	8300

Maximum salary in `EMPLOYEES` table

MAX(SALARY)
24000

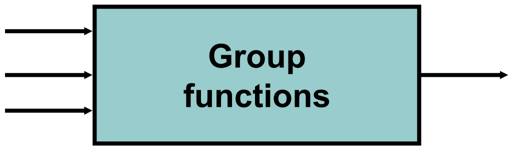
5 - 4

What Are Group Functions?

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

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE



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

Group Functions: Syntax

```
SELECT      group_function(column), ...
FROM        table
[WHERE      condition]
[ORDER BY   column];
```


Group Functions: Syntax

The group function is placed after the `SELECT` keyword. You may have multiple group functions separated by commas.

Guidelines for using the 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.

Using the AVG and SUM Functions

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%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

Using the AVG and SUM Functions

You can use the AVG, SUM, MIN, and MAX functions against the columns that can store numeric data. The example in the slide displays the average, highest, lowest, and sum of monthly salaries for all sales representatives.

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Using the MIN and MAX Functions

You can use the MAX and MIN functions for numeric, character, and date data types. The example in the slide displays the most junior and most senior employees.

The following example displays the employee last name that is first and the employee last name that is last in an alphabetic list of all employees:

```
SELECT MIN(last_name), MAX(last_name)
FROM   employees;
```

	MIN(LAST_NAME)	MAX(LAST_NAME)
1	Abel	Zlotkey

Note: The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types. MAX and MIN cannot be used with LOB or LONG data types.

Using the COUNT Function

COUNT(*) returns the number of rows in a table:

1

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

COUNT(*expr*) returns the number of rows with non-null values for *expr*:

2

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

ORACLE

Using 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 satisfy the condition in the WHERE clause.

In contrast, COUNT(*expr*) returns the number of non-null values that are in the column identified by *expr*.

COUNT(DISTINCT *expr*) returns the number of unique, non-null values that are in the column identified by *expr*.

Examples:

1. The example in the slide displays the number of employees in department 50.
2. The example in the slide displays the number of employees in department 80 who can earn a commission.

Using the `DISTINCT` Keyword

- `COUNT(DISTINCT expr)` returns the number of distinct non-null values of *`expr`*.
- To display the number of distinct department values in the `EMPLOYEES` table:

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

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

Using the `DISTINCT` Keyword

Use the `DISTINCT` keyword to suppress the counting of any duplicate values in a column.

The example in the slide displays the number of distinct department values that are in the `EMPLOYEES` table.

Group Functions and Null Values

Group functions ignore null values in the column:

1

```
SELECT AVG(commission_pct)
FROM   employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

The NVL function forces group functions to include null values:

2

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

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

ORACLE

Group Functions and Null Values

All group functions ignore null values in the column.

However, the NVL function forces group functions to include null values.

Examples:

1. The average is calculated based on *only* those rows in the table in which a valid value is stored in the COMMISSION_PCT column. The average is calculated as the total commission that is paid to all employees divided by the number of employees receiving a commission (four).
2. 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 that is paid to all employees divided by the total number of employees in the company (20).

Lesson Agenda

- Group functions:
  - Types and syntax
  - Use AVG, SUM, MIN, MAX, COUNT
  - Use DISTINCT keyword within group functions
  - NULL values in a group function
- Grouping rows:
  - GROUP BY clause
  - HAVING clause
- Nesting group functions

Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600

...

19	110	12000
20	(null)	7000

4400

9500

3500

6400

10033

Average salary in EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.333333333333...
6	90	19333.333333333333...
7	110	10150
8	(null)	7000

ORACLE

Creating Groups of Data

Until this point in our discussion, all group functions have treated the table as one large group of information. At times, however, you need to divide the table of information into smaller groups. This can be done by using the `GROUP BY` clause.

Creating Groups of Data: GROUP BY Clause Syntax

```
SELECT     column, group_function(column)
FROM       table
[WHERE     condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

You can divide rows in a table into smaller groups by using the GROUP BY clause.

Creating Groups of Data: GROUP BY Clause Syntax

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.

Using the GROUP BY Clause

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

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

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.33333333333...
3	20	9500
4	110	10150
5	50	3500
6	80	10033.33333333333...
7	60	6400
8	10	4400

ORACLE

Using the GROUP BY Clause

When using the GROUP BY clause, make sure that all columns in the SELECT list that are not group functions are included in the GROUP BY clause. The example in the slide displays the department number and the average salary for each department. Here is how this SELECT statement, containing a GROUP BY clause, is evaluated:

- The SELECT clause specifies the columns to be retrieved, as follows:
  - Department number column in the EMPLOYEES table
  - The average of all 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 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 grouped by department number, so the AVG function that is applied to the salary column calculates the average salary for each department.

Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

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

	AVG(SALARY)
1	7000
2	19333.3333333333333333333...
3	9500
4	10150
5	3500
6	10033.3333333333333333333...
7	6400
8	4400

Using the GROUP BY Clause (continued)

The GROUP BY column does not have to be in the SELECT clause. For example, the SELECT statement in the slide 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 also use the group function in the ORDER BY clause:

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

	DEPARTMENT_ID	AVG(SALARY)
1	50	3500
2	10	4400
3	60	6400
...		
7	110	10150
8	90	19333.3333333333333333333333333333333

Grouping by More than One Column

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_MAN	5800
5	50	ST_CLERK	2500
6	50	ST_CLERK	2600
7	50	ST_CLERK	3100
8	50	ST_CLERK	3500
9	60	IT_PROG	4200
10	60	IT_PROG	6000
11	60	IT_PROG	9000
12	80	SA_REP	11000
13	80	SA_MAN	10500
14	80	SA_REP	8600
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Add the salaries in the EMPLOYEES table for each job, grouped by department.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

ORACLE

5 - 17

Grouping by More than One Column

Sometimes you need to see results for groups within groups. The slide shows a report that displays the total salary that is paid to each job title in each department.

The EMPLOYEES table is grouped first by the department number and then by the job title within that grouping. For example, the four stock clerks in department 50 are grouped together, and a single result (total salary) is produced for all stock clerks in the group.

Using the GROUP BY Clause on Multiple Columns

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

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

ORACLE

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. In the example in the slide, the SELECT statement containing a GROUP BY clause is evaluated as follows:

- 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 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 the department number.
  - Second, the rows are grouped by job ID in the department number groups.

So the SUM function is applied to the salary column for all job IDs in each department number group.

Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause:

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

```
ORA-00937: not a single-group group function
00937. 00000 -  "not a single-group group function"
```

A GROUP BY clause must be added to count the last names for each department_id.

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

```
ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
```

Either add job_id in the GROUP BY or remove the job_id column from the SELECT list.

Illegal Queries Using Group Functions

Whenever you use a mixture of individual items (DEPARTMENT_ID) and group functions (COUNT) in the same SELECT statement, you must include a GROUP BY clause that specifies the individual items (in this case, DEPARTMENT_ID). If the GROUP BY clause is missing, then the error message "not a single-group group function" appears and an asterisk (*) points to the offending column. You can correct the error in the first example in the slide by adding the GROUP BY clause:

```
SELECT   department_id, count(last_name)
FROM     employees
GROUP BY department_id;
```

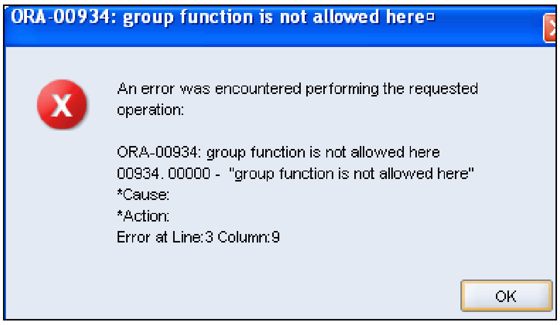
Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause. In the second example in the slide, job_id is neither in the GROUP BY clause nor is it being used by a group function, so there is a "not a GROUP BY expression" error. You can correct the error in the second slide example by adding job_id in the GROUP BY clause.

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id, job_id;
```

Illegal Queries
Using Group Functions

- 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.

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



Cannot use the `WHERE` clause to restrict groups

5 - 20

Illegal Queries Using Group Functions (continued)

The `WHERE` clause cannot be used to restrict groups. The `SELECT` statement in the example in the slide results in an error because it uses the `WHERE` clause to restrict the display of the average salaries of those departments that have an average salary greater than $8,000.

However, you can correct the error in the example by using the `HAVING` clause to restrict groups:

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

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.3333333333333333...
2	20	9500
3	110	10150
4	80	10033.3333333333333333...

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600

...

18	110	8300
19	110	12000
20	(null)	7000

The maximum salary per department when it is greater than $10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	80	11000
3	90	24000
4	110	12000

5 - 21

Restricting Group Results

You use the `HAVING` clause to restrict groups in the same way that you use the `WHERE` clause to restrict the rows that you select. To find the maximum salary in each of the departments that have a maximum salary greater than $10,000, you need to do the following:

1. Find the average salary for each department by grouping by department number.
2. Restrict the groups to those departments with a maximum salary greater than $10,000.

Restricting Group Results with the HAVING Clause

When you use the HAVING clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

5 - 22

Restricting Group Results with the HAVING Clause

You use the HAVING clause to specify the groups that are to be displayed, thus further restricting the groups on the basis of aggregate information.

In the syntax, *group_condition* restricts the groups of rows returned to those groups for which the specified condition is true.

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 it is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

Using the HAVING Clause

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY department_id
HAVING    MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12000
4	80	11000

Using the HAVING Clause

The example in the slide displays the department numbers and maximum salaries for those departments with a maximum salary greater than $10,000.

You can use the GROUP BY clause without using a group function in the SELECT list. If you restrict rows based on the result of a group function, you must have a GROUP BY clause as well as the HAVING clause.

The following example displays the department numbers and average salaries for those departments with a maximum salary greater than $10,000:

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id
HAVING    max(salary)>10000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.3333333333333333...
2	20	9500
3	110	10150
4	80	10033.3333333333333333...

Using the HAVING Clause

```
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);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

5 - 24

Using the HAVING Clause (continued)

The example in the slide displays the job ID and total monthly salary for each job that has a total payroll exceeding $13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

Lesson Agenda

- Group functions:
  - Types and syntax
  - Use `AVG`, `SUM`, `MIN`, `MAX`, `COUNT`
  - Use `DISTINCT` keyword within group functions
  - `NULL` values in a group function
- Grouping rows:
  - `GROUP BY` clause
  - `HAVING` clause
- Nesting group functions

5 - 25

Nesting Group Functions

Display the maximum average salary:

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

	MAX(AVG(SALARY))
1	19333.3333333333333333333333333333333333

Nesting Group Functions

Group functions can be nested to a depth of two functions. The example in the slide calculates the average salary for each `department_id` and then displays the maximum average salary.

Note that `GROUP BY` clause is mandatory when nesting group functions.

Summary

In this lesson, you should have learned how to:

- Use the group functions COUNT, MAX, MIN, SUM, and AVG
- Write queries that use the GROUP BY clause
- Write queries that use the HAVING clause

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```


Summary

There are several group functions available in SQL, such as:
AVG, COUNT, MAX, MIN, SUM, STDDEV, and VARIANCE

You can create subgroups by using the GROUP BY clause. Further, groups can be restricted using the HAVING clause.

Place the HAVING and GROUP BY clauses after the WHERE clause in a statement. The order of the GROUP BY and HAVING clauses following the WHERE clause is not important. Place the ORDER BY clause at the end.

The Oracle server evaluates the clauses in the following order:

1. If the statement contains a WHERE clause, the server establishes the candidate rows.
2. The server identifies the groups that are specified in the GROUP BY clause.
3. The HAVING clause further restricts result groups that do not meet the group criteria in the HAVING clause.

Note: For a complete list of the group functions, see *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Practice 5: Overview

This practice covers the following topics:

- Writing queries that use the group functions
- Grouping by rows to achieve more than one result
- Restricting groups by using the HAVING clause

Practice 5: Overview

At the end of this practice, you should be familiar with using group functions and selecting groups of data.

Practice 5

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.
   True/False

2. Group functions include nulls in calculations.
   True/False

3. The WHERE clause restricts rows before inclusion in a group calculation.
   True/False

The HR department needs the following reports:

4. Find the highest, lowest, sum, and average salary of all employees. Label the columns as Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Save your SQL statement as lab_05_04.sql. Run the query.

	Maximum	Minimum	Sum	Average
1	24000	2500	175500	8775

5. Modify the query in lab_05_04.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab_05_04.sql as lab_05_05.sql. Run the statement in lab_05_05.sql.

	JOB_ID	Maximum	Minimum	Sum	Average
1	IT_PROG	9000	4200	19200	6400
2	AC_MGR	12000	12000	12000	12000
3	AC_ACCOUNT	8300	8300	8300	8300
4	ST_MAN	5800	5800	5800	5800
5	AD_ASST	4400	4400	4400	4400
6	AD_VP	17000	17000	34000	17000
7	SA_MAN	10500	10500	10500	10500
8	MK_MAN	13000	13000	13000	13000
9	AD_PRES	24000	24000	24000	24000
10	SA_REP	11000	7000	26600	8867
11	MK_REP	6000	6000	6000	6000
12	ST_CLERK	3500	2500	11700	2925

Practice 5 (continued)

6. Write a query to display the number of people with the same job.

	JOB_ID	COUNT(*)
1	AC_ACCOUNT	1
2	AC_MGR	1
3	AD_ASST	1
4	AD_PRES	1
5	AD_VP	2
6	IT_PROG	3
7	MK_MAN	1
8	MK_REP	1
9	SA_MAN	1
10	SA_REP	3
11	ST_CLERK	4
12	ST_MAN	1

Generalize the query so that the user in the HR department is prompted for a job title. Save the script to a file named `lab_05_06.sql`. Run the query. Enter `IT_PROG` when prompted.

	JOB_ID	COUNT(*)
1	IT_PROG	3

7. Determine the number of managers without listing them. Label the column as `Number of Managers`. *Hint: Use the `MANAGER_ID` column to determine the number of managers.*

	Number of Managers
1	8

8. Find the difference between the highest and lowest salaries. Label the column `DIFFERENCE`.

	DIFFERENCE
1	21500

Practice 5 (continued)

If you have time, complete the following exercises:

9. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.

	MANAGER_ID	MIN(SALARY)
1	102	9000
2	205	8300
3	149	7000

If you want an extra challenge, complete the following exercises:

10. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

	TOTAL	1995	1996	1997	1998
1	20	1	2	2	3

11. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

	Job	Dept 20	Dept 50	Dept 80	Dept 90	Total
1	IT_PROG	(null)	(null)	(null)	(null)	19200
2	AC_MGR	(null)	(null)	(null)	(null)	12000
3	AC_ACCOUNT	(null)	(null)	(null)	(null)	8300
4	ST_MAN	(null)	5800	(null)	(null)	5800
5	AD_ASST	(null)	(null)	(null)	(null)	4400
6	AD_VP	(null)	(null)	(null)	34000	34000
7	SA_MAN	(null)	(null)	10500	(null)	10500
8	MK_MAN	13000	(null)	(null)	(null)	13000
9	AD_PRES	(null)	(null)	(null)	24000	24000
10	SA_REP	(null)	(null)	19600	(null)	26600
11	MK_REP	6000	(null)	(null)	(null)	6000
12	ST_CLERK	(null)	11700	(null)	(null)	11700

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 equijoins and nonequijoins
- Join a table to itself by using a self-join
- View data that generally does not meet a join condition by using outer joins
- Generate a Cartesian product of all rows from two or more tables

6 - 2

Objectives

This lesson explains how to obtain data from more than one table. A *join* is used to view information from multiple tables. Therefore, you can *join* tables together to view information from more than one table.

Note: Information on joins is found in the section on *SQL Queries and Subqueries: Joins* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Types of `JOINS` and its syntax
- Natural join:
  - `USING` clause
  - `ON` clause
- Self-join
- Nonequijoins
- `OUTER` join:
  - `LEFT OUTER` join
  - `RIGHT OUTER` join
  - `FULL OUTER` join
- Cartesian product
  - Cross join

ORACLE

6 - 3

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
…			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
5	144	50	Shipping
…			
18	205	110	Accounting
19	206	110	Accounting

ORACLE

6 - 4

Obtaining Data from Multiple Tables

Sometimes you need to use data from more than one table. In the example in the slide, the report displays data from two separate tables:

- Employee IDs exist in the `EMPLOYEES` table.
- Department IDs exist in both the `EMPLOYEES` and `DEPARTMENTS` tables.
- Department names 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.

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Natural joins:
  - `NATURAL JOIN` clause
  - `USING` clause
  - `ON` clause
- Outer joins:
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`
- Cross joins

6 - 5

Types of Joins

To join tables, you can use a join syntax that is compliant with the SQL:1999 standard.

Note: Before the Oracle9*i* release, the join syntax was different from the American National Standards Institute (ANSI) standards. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in the prior releases. For detailed information about the proprietary join syntax, see Appendix C: Oracle Join Syntax.

Note: The following slide discusses the SQL:1999 join syntax.

Joining Tables Using SQL:1999 Syntax

Use a join to query data from more than one table:

```
SELECT   table1.column, table2.column
FROM     table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

ORACLE

6 - 6

Joining Tables Using SQL:1999 Syntax

In the syntax:

table1.column denotes the table and the column from which data is retrieved

NATURAL JOIN joins two tables based on the same column name

JOIN *table2* USING *column_name* performs an equijoin based on the column name

JOIN *table2* ON *table1.column_name* = *table2.column_name* performs an equijoin based on the condition in the ON clause

LEFT/RIGHT/FULL OUTER is used to perform outer joins

CROSS JOIN returns a Cartesian product from the two tables

For more information, see the section titled *SELECT* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Instead of full table name prefixes, use table aliases.
- Table alias gives a table a shorter name:
  - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

Qualifying Ambiguous Column Names

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the `DEPARTMENT_ID` column in the `SELECT` list 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.

However, qualifying column names with table names can be time consuming, particularly if the table names are lengthy. Instead, you can use *table aliases*. 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, followed by a space and then the table alias. For example, the `EMPLOYEES` table can be given an alias of `e`, and the `DEPARTMENTS` table an alias of `d`.

Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- 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 for only the current `SELECT` statement.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequijoins
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

6 - 9

Creating Natural Joins

You can join tables automatically based on the columns in the two tables that have matching data types and names. You do this by using the `NATURAL JOIN` keywords.

Note: The join can happen on only those columns that have the same names and data types in both tables. If the columns have the same name but different data types, then the `NATURAL JOIN` syntax causes an error.

Retrieving Records with Natural Joins

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

ORACLE

6 - 10

Retrieving Records with Natural Joins

In the example in the slide, the `LOCATIONS` table is joined to the `DEPARTMENT` table by the `LOCATION_ID` column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

Natural Joins with a `WHERE` Clause

Additional restrictions on a natural join are implemented by using a `WHERE` clause. The following example limits the rows of output to those with a department ID equal to 20 or 50:

```
SELECT   department_id, department_name,
         location_id, city
FROM     departments
NATURAL JOIN locations
WHERE    department_id IN (20, 50);
```

Creating Joins with the `USING` Clause

- If several columns have the same names but the data types do not match, natural join can be applied using the `USING` clause to specify the columns that should be used for an equijoin.
- Use the `USING` clause to match only one column when more than one column matches.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

Creating Joins with the `USING` Clause

Natural joins use all columns with matching names and data types to join the tables. The `USING` clause can be used to specify only those columns that should be used for an equijoin.

Joining Column Names

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
100	90
101	90
102	90
103	60
104	60
107	60
124	50
141	50
142	50
143	50
144	50
149	80
174	80
176	80

...

Foreign key

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Primary key

6 - 12

Joining Column Names

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 in both the 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*.

Retrieving Records with the USING Clause

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60
...				
19	205	Higgins	1700	110

ORACLE

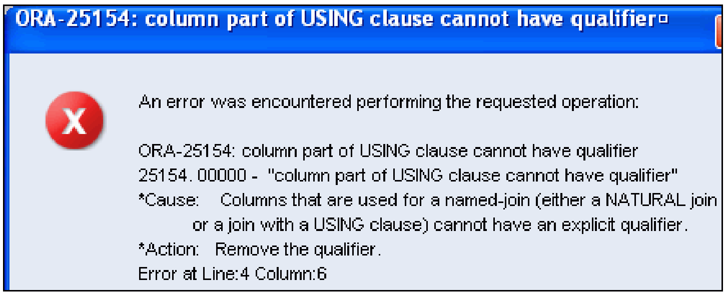
Retrieving Records with the USING Clause

In the example in the slide, the DEPARTMENT_ID columns in the EMPLOYEES and DEPARTMENTS tables are joined and thus the LOCATION_ID of the department where an employee works is shown.

Using Table Aliases with the `USING` Clause

- Do not qualify a column that is used in the `USING` clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE d.location_id = 1400;
```



Using Table Aliases with the `USING` clause

When joining with the `USING` clause, you cannot qualify a column that is used in the `USING` clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it. For example, in the query mentioned in the slide, you should not alias the `location_id` column in the `WHERE` clause because the column is used in the `USING` clause.

The columns that are referenced in the `USING` clause should not have a qualifier (table name or alias) anywhere in the SQL statement. For example, the following statement is valid:

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING (location_id)
WHERE  location_id = 1400;
```

Because, other columns that are common in both the tables, but not used in the `USING` clause, must be prefixed with a table alias otherwise you get the "column ambiguously defined" error.

In the following statement, `manager_id` is present in both the `employees` and `departments` table and if `manager_id` is not prefixed with a table alias, it gives a "column ambiguously defined" error.

The following statement is valid:

```
SELECT first_name, d.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
WHERE  department_id = 50;
```

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

ORACLE

Creating Joins with the ON Clause

Use the ON clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the WHERE clause.

Retrieving Records with the ON Clause

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400

...

Retrieving Records with the ON Clause

In this example, the DEPARTMENT_ID columns in the EMPLOYEES and DEPARTMENTS table are joined using the ON clause. Wherever a department ID in the EMPLOYEES table equals a department ID in the DEPARTMENTS table, the row is returned. The table alias is necessary to qualify the matching column_names.

You can also use the ON clause to join columns that have different names. The parenthesis around the joined columns as in the slide example, (e.department_id = d.department_id) is optional. So, even ON e.department_id = d.department_id will work.

Note: SQL Developer suffixes a '_1' to differentiate between the two department_ids.

Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping

...

Creating Three-Way Joins with the ON Clause

A three-way join is a join of three tables. In SQL:1999–compliant syntax, joins are performed from left to right. So, the first join to be performed is EMPLOYEES JOIN DEPARTMENTS. The first join condition can reference columns in EMPLOYEES and DEPARTMENTS but cannot reference columns in LOCATIONS. The second join condition can reference columns from all three tables.

Note: The code example in the slide can also be accomplished with the USING clause:

```
SELECT e.employee_id, l.city, d.department_name
FROM employees e
JOIN departments d
USING (department_id)
JOIN locations l
USING (location_id)
```

Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply additional conditions:

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

Or

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
WHERE   e.manager_id = 149 ;
```

ORACLE

6 - 18

Applying Additional Conditions to a Join

You can apply additional conditions to the join.

The example shown performs a join on the EMPLOYEES and DEPARTMENTS tables and, in addition, displays only employees who have a manager ID of 149. To add additional conditions to the ON clause, you can add AND clauses. Alternatively, you can use a WHERE clause to apply additional conditions.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

Lesson Agenda

- Types of `JOINS` and its syntax
- Natural join:
  - `USING` clause
  - `ON` clause
- Self-join
- Nonequijoins
- `OUTER` join:
  - `LEFT OUTER` join
  - `RIGHT OUTER` join
  - `FULL OUTER` join
- Cartesian product
  - Cross join

Joining a Table to Itself

EMPLOYEES (WORKER)

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

…

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos
141	Rajs
142	Davies
143	Matos

…

MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

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. For example, to find the name of Lorentz's manager, you need to:

- Find Lorentz in the EMPLOYEES table by looking at the LAST_NAME column
- Find the manager number for Lorentz by looking at the MANAGER_ID column. Lorentz's manager number is 103.
- Find the name of the manager with EMPLOYEE_ID 103 by looking at the LAST_NAME column. Hunold's employee number is 103, so Hunold is Lorentz's manager.

In this process, you look in the table twice. The first time you look in the table to find Lorentz in the LAST_NAME column and the MANAGER_ID value of 103. The second time you look in the EMPLOYEE_ID column to find 103 and the LAST_NAME column to find Hunold.

Self-Joins Using the ON Clause

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON     (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

...

ORACLE

6 - 21

Self-Joins Using the ON Clause

The ON clause can also be used to join columns that have different names, within the same table or in a different table.

The example shown is a self-join of the EMPLOYEES table, based on the EMPLOYEE_ID and MANAGER_ID columns.

Note: The parenthesis around the joined columns as in the slide example, (e.manager_id = m.employee_id) is **optional**. So, even ON e.manager_id = m.employee_id will work.

Lesson Agenda

- Types of `JOINS` and its syntax
- Natural join:
  - `USING` clause
  - `ON` clause
- Self-join
- Nonequijoins
- `OUTER` join:
  - `LEFT OUTER` join
  - `RIGHT OUTER` join
  - `FULL OUTER` join
- Cartesian product
  - Cross join

Nonequijoins

EMPLOYEES

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hunold	9000
5	Ernst	6000
6	Lorentz	4200
7	Mourgos	5800
8	Rajs	3500
9	Davies	3100
10	Matos	2600
...		
19	Higgins	12000
20	Gietz	8300

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

`JOB_GRADES` table defines the `LOWEST_SAL` and `HIGHEST_SAL` range of values for each `GRADE_LEVEL`. Hence, the `GRADE_LEVEL` column can be used to assign grades to each employee.

6 - 23

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 is an example of a nonequijoin. The `SALARY` column in the `EMPLOYEES` table ranges between the values in the `LOWEST_SAL` and `HIGHEST_SAL` columns of the `JOB_GRADES` table. Hence, each employee can be graded based on their salary. The relationship is obtained using an operator other than the equality (=) operator.

Retrieving Records with Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

ORACLE

Retrieving Records with Nonequijoins

The slide example 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_GRADES` table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values 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 the `BETWEEN` condition. The Oracle server translates the `BETWEEN` condition to a pair of `AND` conditions. Therefore, using `BETWEEN` has no performance benefits, but should be used only for logical simplicity.

Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- Nonequijoins
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join
- Cartesian product
  - Cross join

Returning Records with No Direct Match with Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	90	King
2	90	Kochhar
3	90	De Haan
4	60	Hunold
5	60	Ernst
6	60	Lorentz
7	50	Mourgos
8	50	Rajs
9	50	Davies
10	50	Matos
...		
19	110	Higgins
20	110	Gietz

There are no employees in department 190.

6 - 26

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of `EMPLOYEES` and `DEPARTMENTS` tables, department ID 190 does not appear because there are no employees with that department ID recorded in the `EMPLOYEES` table. Therefore, instead of seeing 20 employees in the result set, you see 19 records.

To return the department record that does not have any employees, you can use an outer join.

`INNER` Versus `OUTER` Joins

- In SQL:1999, the join of two tables returning only matched rows is called an inner join.
- A join between two tables that returns the results of the inner join as well as the unmatched rows from the left (or right) table is called a left (or right) outer join.
- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

ORACLE

`INNER` Versus `OUTER` Joins

Joining tables with the `NATURAL JOIN`, `USING`, or `ON` clauses results in an inner join. Any unmatched rows are not displayed in the output. To return the unmatched rows, you can use an outer join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition.

There are three types of outer joins:

- `LEFT OUTER`
- `RIGHT OUTER`
- `FULL OUTER`

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

…

17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

LEFT OUTER JOIN

This query retrieves all rows in the EMPLOYEES table, which is the left table, even if there is no match in the DEPARTMENTS table.

RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

ORACLE

RIGHT OUTER JOIN

This query retrieves all rows in the DEPARTMENTS table, which is the right table, even if there is no match in the EMPLOYEES table.

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

FULL OUTER JOIN

This query retrieves all rows in the `EMPLOYEES` table, even if there is no match in the `DEPARTMENTS` table. It also retrieves all rows in the `DEPARTMENTS` table, even if there is no match in the `EMPLOYEES` table.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
  – USING clause
  – ON clause
- Self-join
- Nonequiijoin
- OUTER join:
  – LEFT OUTER join
  – RIGHT OUTER join
  – FULL OUTER join
- Cartesian product
  – Cross join

Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition.

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 the result is rarely useful. You should, therefore, always include a valid join condition unless you have a specific need to combine all rows from all tables.

However, Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

Generating a Cartesian Product

EMPLOYEES (20 rows)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60
…			
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS (8 rows)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Cartesian product: 20 x 8 = 160 rows

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	100	90	1700
2	101	90	1700
3	102	90	1700
4	103	60	1700
…			
159	205	110	1700
160	206	110	1700

Generating a Cartesian Product

A Cartesian product is generated if a join condition is omitted. The example in the slide displays the employee last name and the department name from the `EMPLOYEES` and `DEPARTMENTS` tables. Because no join condition was specified, 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.

Creating Cross Joins

- The CROSS JOIN clause produces the cross-product of two tables.
- This is also called a Cartesian product between the two tables.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration

...

159	Whalen	Contracting
160	Zlotkey	Contracting

ORACLE

Creating Cross Joins

The example in the slide produces a Cartesian product of the EMPLOYEES and DEPARTMENTS tables.

Summary

In this lesson, you should have learned how to use joins to display data from multiple tables by using:

- Equijoins
- Nonequijoins
- Outer joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) outer joins

Summary

There are multiple ways to join tables.

Types of Joins

- Equijoins
- Nonequijoins
- Outer joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) outer joins

Cartesian Products

A Cartesian product results in the display of all combinations of rows. This is done by either omitting the `WHERE` clause or by specifying the `CROSS JOIN` clause.

Table Aliases

- Table aliases speed up database access.
- Table aliases can help to keep SQL code smaller by conserving memory.
- Table aliases are sometimes mandatory to avoid column ambiguity.

Practice 6: Overview

This practice covers the following topics:

- Joining tables using an equijoin
- Performing outer and self-joins
- Adding conditions

Practice 6: Overview

This practice is intended to give you experience in extracting data from more than one table using the SQL:1999–compliant joins.

Practice 6

1. Write a query for the HR department to produce the addresses of all the departments. Use the `LOCATIONS` and `COUNTRIES` tables. Show the location ID, street address, city, state or province, and country in the output. Use a `NATURAL JOIN` to produce the results.

	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
2	1500	2011 Interiors Blvd	South San Francisco	California	United States of America
3	1700	2004 Charade Rd	Seattle	Washington	United States of America
4	1800	460 Bloor St. W.	Toronto	Ontario	Canada
5	2500	Magdalen Centre, The ...	Oxford	Oxford	United Kingdom

2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all the employees.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping
9	Hunold	60	IT
10	Ernst	60	IT

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting

Practice 6 (continued)

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and the department name for all employees who work in Toronto.

	LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	Hartstein	MK_MAN	20	Marketing
2	Fay	MK_REP	20	Marketing

4. Create a report to display employees' last name and employee number along with their manager's last name and manager number. Label the columns `Employee`, `Emp#`, `Manager`, and `Mgr#`, respectively. Save your SQL statement as `lab_06_04.sql`. Run the query.

	Employee	EMP#	Manager	Mgr#
1	Kochhar	101	King	100
2	De Haan	102	King	100
3	Hunold	103	De Haan	102
4	Ernst	104	Hunold	103
5	Lorentz	107	Hunold	103
6	Mourgos	124	King	100
7	Rajs	141	Mourgos	124
8	Davies	142	Mourgos	124
9	Matos	143	Mourgos	124
10	Vargas	144	Mourgos	124

…

15	Whalen	200	Kochhar	101
16	Hartstein	201	King	100
17	Fay	202	Hartstein	201
18	Higgins	205	Kochhar	101
19	Gietz	206	Higgins	205

Practice 6 (continued)

5. Modify `lab_06_04.sql` to display all employees including King, who has no manager. Order the results by the employee number. Save your SQL statement as `lab_06_05.sql`. Run the query in `lab_06_05.sql`.

	Employee	EMP#	Manager	Mgr#
1	King	100	(null)	(null)
2	Kochhar	101	King	100
3	De Haan	102	King	100
4	Hunold	103	De Haan	102
5	Ernst	104	Hunold	103
6	Lorentz	107	Hunold	103
7	Mourgos	124	King	100
8	Rajs	141	Mourgos	124
9	Davies	142	Mourgos	124
10	Matos	143	Mourgos	124
...				
18	Fay	202	Hartstein	201
19	Higgins	205	Kochhar	101
20	Gietz	206	Higgins	205

6. Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_06_06.sql`.

	DEPARTMENT	EMPLOYEE	COLLEAGUE
1	20	Fay	Hartstein
2	20	Hartstein	Fay
3	50	Davies	Matos
4	50	Davies	Mourgos
5	50	Davies	Rajs
6	50	Davies	Vargas
7	50	Matos	Davies
8	50	Matos	Mourgos
9	50	Matos	Rajs
10	50	Matos	Vargas
...			
42	110	Higgins	Gietz

Practice 6 (continued)

7. The HR department needs a report on job grades and salaries. To familiarize yourself with the `JOB_GRADES` table, first show the structure of the `JOB_GRADES` table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES
Name                           Null     Type
------------------------------ -------- --------------
GRADE_LEVEL                             VARCHAR2(3)
LOWEST_SAL                              NUMBER
HIGHEST_SAL                             NUMBER

3 rows selected
```

	LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
1	Vargas	ST_CLERK	Shipping	2500	A
2	Matos	ST_CLERK	Shipping	2600	A
3	Davies	ST_CLERK	Shipping	3100	B
4	Rajs	ST_CLERK	Shipping	3500	B
5	Lorentz	IT_PROG	IT	4200	B
6	Whalen	AD_ASST	Administration	4400	B
7	Mourgos	ST_MAN	Shipping	5800	B
8	Ernst	IT_PROG	IT	6000	C
9	Fay	MK_REP	Marketing	6000	C
10	Gietz	AC_ACCOUNT	Accounting	8300	C
...					
18	De Haan	AD_VP	Executive	17000	E
19	King	AD_PRES	Executive	24000	E

Practice 6 (continued)

If you want an extra challenge, complete the following exercises:

8. The HR department wants to determine the names of all the employees who were hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

	LAST_NAME	HIRE_DATE
1	Lorentz	07-FEB-99
2	Mourgos	16-NOV-99
3	Matos	15-MAR-98
4	Vargas	09-JUL-98
5	Zlotkey	29-JAN-00
6	Taylor	24-MAR-98
7	Grant	24-MAY-99
8	Fay	17-AUG-97

9. The HR department needs to find the names and hire dates of all the employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab_06_09.sql`.

	LAST_NAME	HIRE_DATE	LAST_NAME_1	HIRE_DATE_1
1	Whalen	17-SEP-87	Kochhar	21-SEP-89
2	Hunold	03-JAN-90	De Haan	13-JAN-93
3	Vargas	09-JUL-98	Mourgos	16-NOV-99
4	Matos	15-MAR-98	Mourgos	16-NOV-99
5	Davies	29-JAN-97	Mourgos	16-NOV-99
6	Rajs	17-OCT-95	Mourgos	16-NOV-99
7	Grant	24-MAY-99	Zlotkey	29-JAN-00
8	Taylor	24-MAR-98	Zlotkey	29-JAN-00
9	Abel	11-MAY-96	Zlotkey	29-JAN-00