SQL1

Student Guide • Volume 2

40057GC10
Production 1.0
July 2001
D33479

ORACLE®

Authors

Nancy Greenberg
Priya Nathan

Technical Contributors and Reviewers

Josephine Turner
Martin Alvarez
Anna Atkinson
Don Bates
Marco Berbeek
Andrew Brannigan
Laszlo Czinkoczki
Michael Gerlach
Sharon Gray
Rosita Hanoman
Mozhe Jalali
Sarah Jones
Charbel Khouri
Christopher Lawless
Diana Lorentz
Nina Minchen
Cuong Nguyen
Daphne Nougier
Patrick Odell
Laura Pezzini
Stacey Procter
Maribel Renau
Bryan Roberts
Helen Robertson
Sunshine Salmon
Casa Sharif
Bernard Soleillant
Craig Spoonemore
Ruediger Steffan
Karla Villasenor
Andree Wheeley
Lachlan Williams

Publisher

Sheryl Domingue

Contents

13

Controlling User Access

Objectives

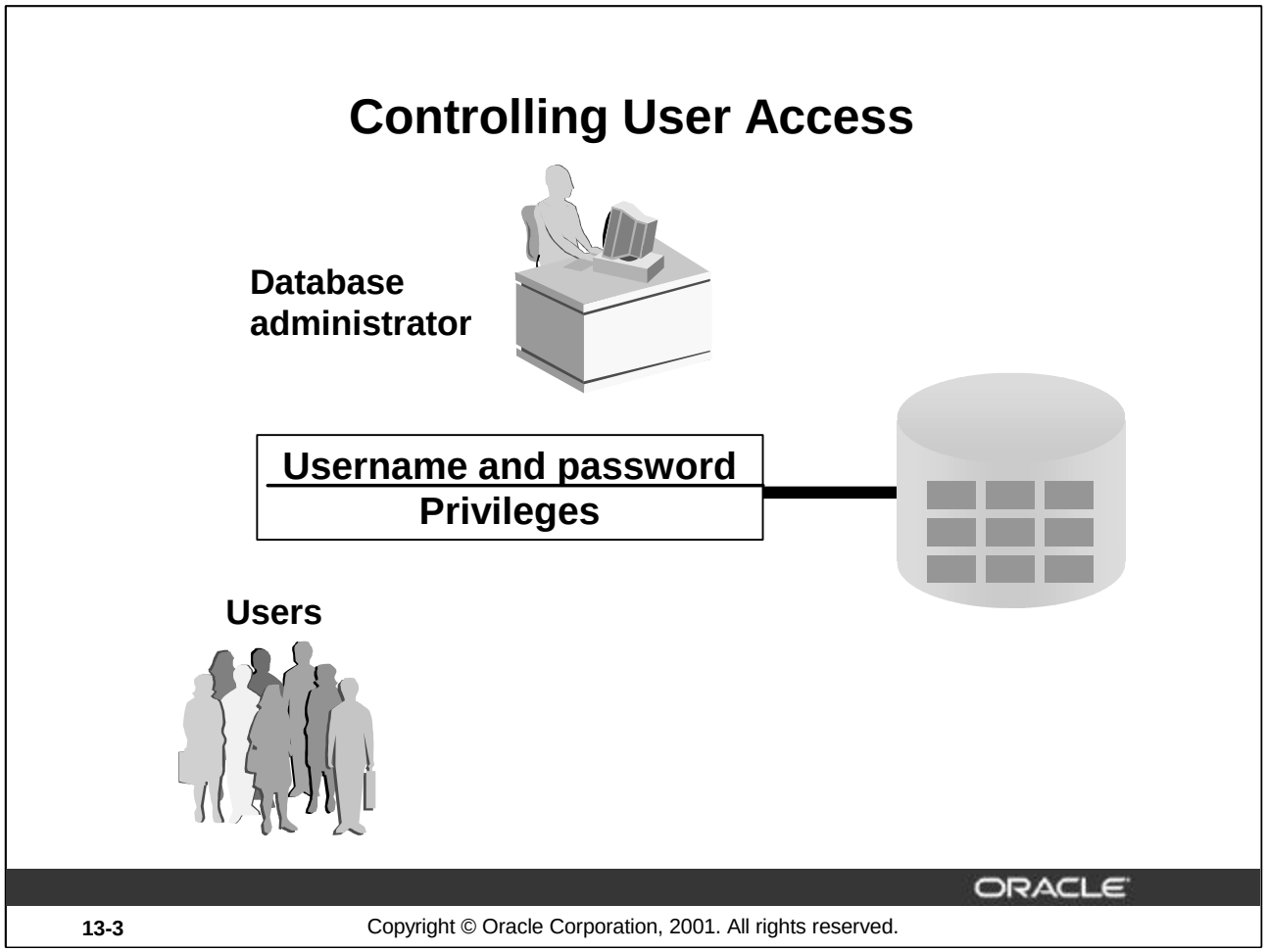
After completing this lesson, you should be able to do the following:

- **Create users**
- **Create roles to ease setup and maintenance of the security model**
- **Use the `GRANT` and `REVOKE` statements to grant and revoke object privileges**
- **Create and access database links**

ORACLE

Lesson Aim

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



Controlling User Access

In a multiple-user environment, you want to maintain security of the database access and use. With Oracle server database security, you can do the following:

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

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

Privileges

- **Database security:**
  - **System security**
  - **Data security**
- **System privileges: Gaining access to the database**
- **Object privileges: Manipulating the content of the database objects**
- **Schemas: Collections of objects, such as tables, views, and sequences**

ORACLE

Privileges

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

Schemas

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

For more information, see *Oracle9i Application Developer's Guide - Fundamentals*, "Establishing a Security Policy" section, and *Oracle9i Concepts*, "Database Security" topic.

System Privileges

- More than 100 privileges are available.
- The database administrator has high-level system privileges for tasks such as:
  - Creating new users
  - Removing users
  - Removing tables
  - Backing up tables

ORACLE

System Privileges

More than 100 distinct system privileges are available for users and roles. System privileges typically are provided by the database administrator.

Typical DBA Privileges

System Privilege	Operations Authorized
CREATE USER	Grantee can create other Oracle users (a privilege required for a DBA role).
DROP USER	Grantee can drop another user.
DROP ANY TABLE	Grantee can drop a table in any schema.
BACKUP ANY TABLE	Grantee can back up any table in any schema with the export utility.
SELECT ANY TABLE	Grantee can query tables, views, or snapshots in any schema.
CREATE ANY TABLE	Grantee can create tables in any schema.

Creating Users

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

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

```
CREATE USER  scott
IDENTIFIED BY   tiger;
User created.
```

ORACLE

Creating a User

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

The slide gives the abridged syntax for creating a user.

In the syntax:

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

For more information, see *Oracle9i SQL Reference*, "`GRANT`" and "`CREATE USER`."

User System Privileges

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

```
GRANT privilege [, privilege...]
TO user [, user| role, PUBLIC...];
```

- **An application developer, for example, may have the following system privileges:**
  - **CREATE SESSION**
  - **CREATE TABLE**
  - **CREATE SEQUENCE**
  - **CREATE VIEW**
  - **CREATE PROCEDURE**

ORACLE

Typical User Privileges

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

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

In the syntax:

privilege	is the system privilege to be granted
user |role|PUBLIC	is the name of the user, the name of the role, or PUBLIC designates that every user is granted the privilege

Note: Current system privileges can be found in the dictionary view SESSION_PRIVS.

Granting System Privileges

The DBA can grant a user specific system privileges.

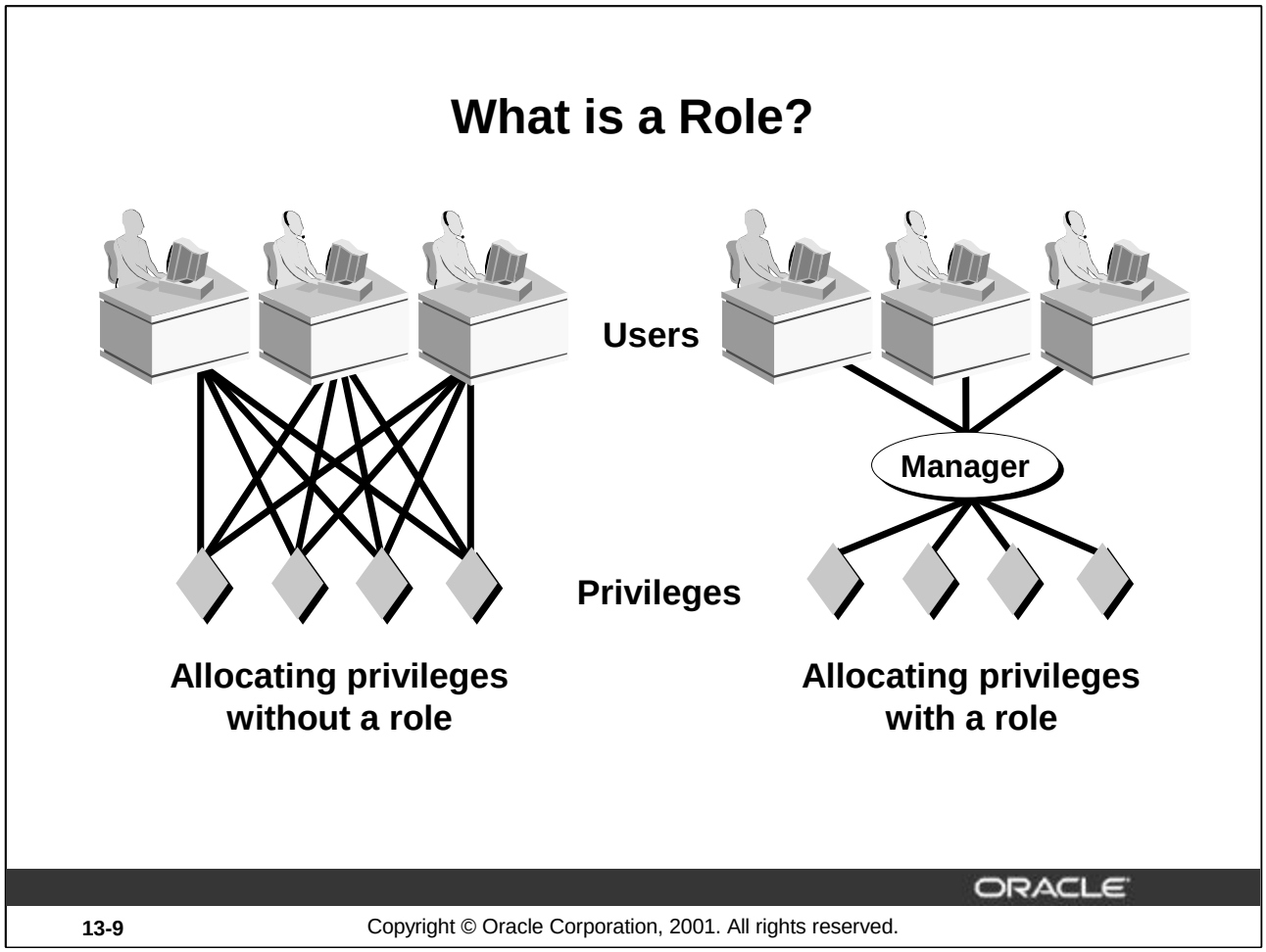
```
GRANT  create session, create table,
       create sequence, create view
TO     scott;
Grant succeeded.
```

ORACLE

Granting System Privileges

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

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



What is a Role?

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

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

Creating and Assigning a Role

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

Syntax

```
CREATE   ROLE  role;
```

In the syntax:

role is the name of the role to be created

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

Creating and Granting Privileges to a Role

- Create a role

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

- Grant privileges to a role

```
GRANT create table, create view
TO manager;
Grant succeeded.
```

- Grant a role to users

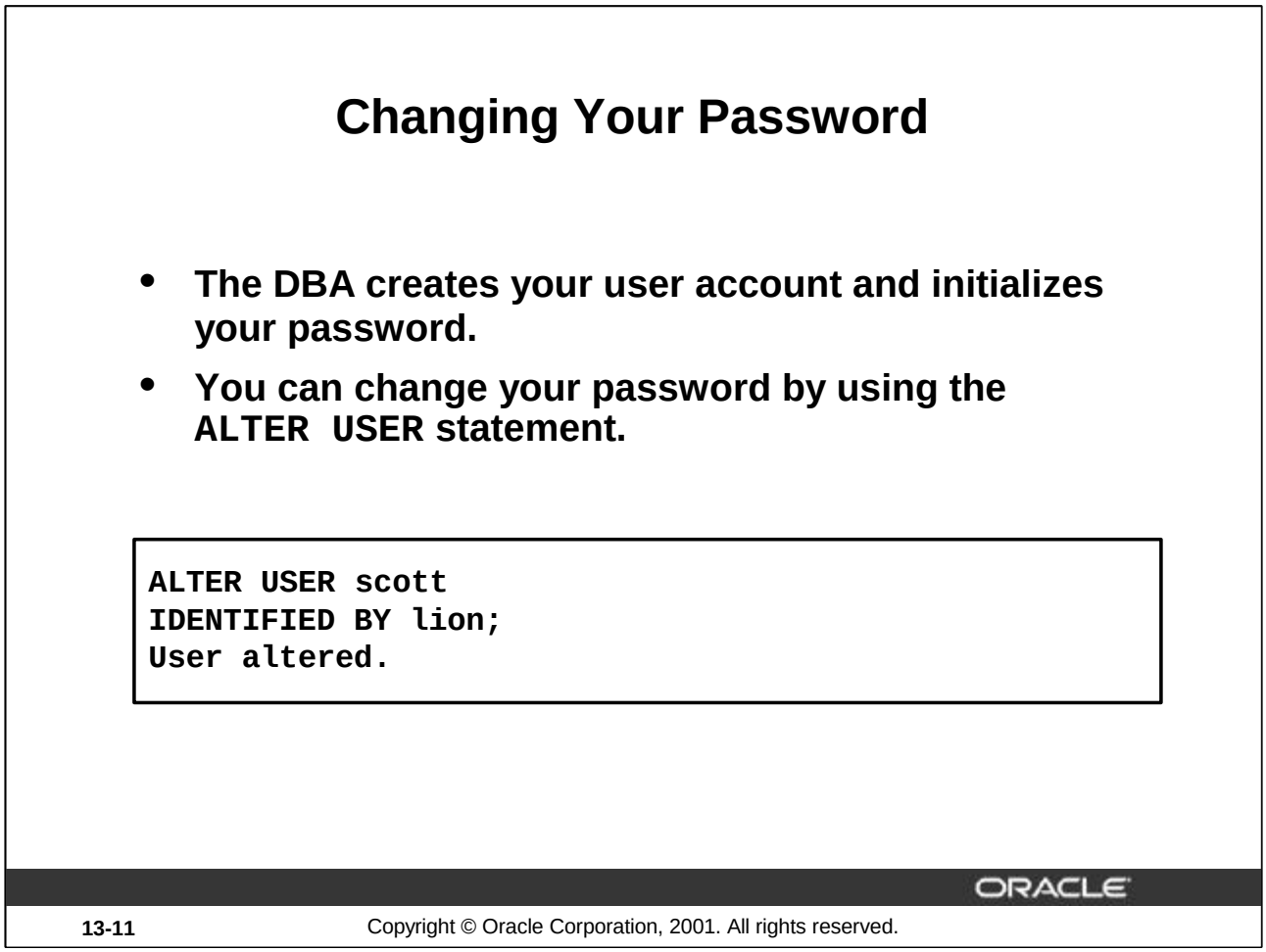
```
GRANT manager TO DEHAAN, KOCHHAR;
Grant succeeded.
```

ORACLE

Creating a Role

The example on the slide creates a manager role and then allows managers to create tables and views. It then grants DeHaan and Kochhar the role of managers. Now DeHaan and Kochhar can create tables and views.

If users have multiple roles granted to them, they receive all of the privileges associated with all of the roles.



Changing Your Password

The DBA creates an account and initializes a password for every user. You can change your password by using the ALTER USER statement.

Syntax

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

In the syntax:

user	is the name of the user
password	specifies the new password

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

For more information, see *Oracle9i SQL Reference*, "ALTER USER."

Object Privileges

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

ORACLE

Object Privileges

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

Object Privileges

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

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

ORACLE

Granting Object Privileges

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

In the syntax:

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

Granting Object Privileges

- **Grant query privileges on the EMPLOYEES table.**

```
GRANT  select
ON     employees
TO     sue, rich;
Grant succeeded.
```

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

```
GRANT  update (department_name, location_id)
ON     departments
TO     scott, manager;
Grant succeeded.
```

ORACLE

Guidelines

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

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

If Sue or Rich now want to SELECT data from the employees table, the syntax they must use is:

```
SELECT  *
FROM    scott.employees;
```

Alternatively, they can create a synonym for the table and SELECT from the synonym:

```
CREATE SYNONYM emp FOR scott.employees;
SELECT * FROM emp;
```

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

Using the WITH GRANT OPTION and PUBLIC Keywords

- Give a user authority to pass along privileges.

```
GRANT  select, insert
ON     departments
TO     scott
WITH   GRANT OPTION;
Grant succeeded.
```

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

```
GRANT  select
ON     alice.departments
TO     PUBLIC;
Grant succeeded.
```

ORACLE

The WITH GRANT OPTION Keyword

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

The example on the slide gives user Scott access to your DEPARTMENTS table with the privileges to query the table and add rows to the table. The example also allows Scott to give others these privileges.

The PUBLIC Keyword

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

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

Confirming Privileges Granted

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

ORACLE

Confirming Granted Privileges

If you attempt to perform an unauthorized operation, such as deleting a row from a table for which you do not have the DELETE privilege, the Oracle server does not permit the operation to take place.

If you receive the Oracle server error message "table or view does not exist," you have done either of the following:

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

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

How to Revoke Object Privileges

- **You use the `REVOKE` statement to revoke privileges granted to other users.**
- **Privileges granted to others through the `WITH GRANT OPTION` clause are also revoked.**

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

Revoking Object Privileges

You can remove privileges granted to other users by using the `REVOKE` statement. When you use the `REVOKE` statement, the privileges that you specify are revoked from the users you name and from any other users to whom those privileges were granted through the `WITH GRANT OPTION` clause.

In the syntax:

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

For more information, see *Oracle9i SQL Reference*, "`REVOKE`."

Revoking Object Privileges

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

```
REVOKE  select, insert
ON      departments
FROM    scott;
Revoke succeeded.
```

ORACLE

Revoking Object Privileges (continued)

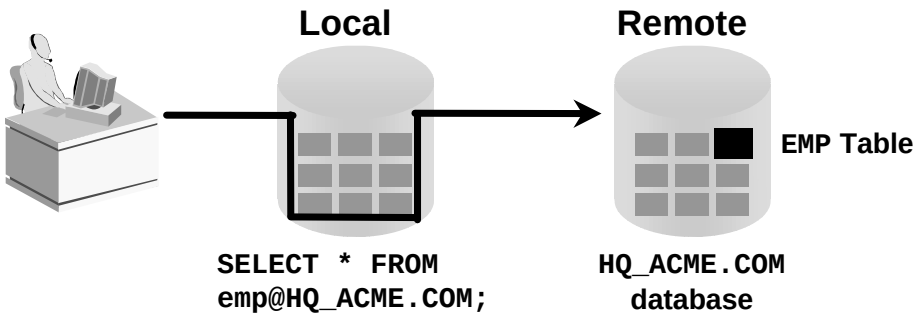
The example on the slide revokes SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

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

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

Database Links

A database link connection allows local users to access data on a remote database.



ORACLE

13-19

Database Links

A database link is a pointer that defines a one-way communication path from an Oracle database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry.

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, they must define a link that is stored in the data dictionary of database B.

A database link connection gives local users access to data on a remote database. For this connection to occur, each database in the distributed system must have a unique global database name. The global database name uniquely identifies a database server in a distributed system.

The great advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object's owner. In other words, a local user can access a remote database without having to be a user on the remote database.

The example shows a user `SCOTT` accessing the `EMP` table on the remote database with the global name `HQ.ACME.COM`.

Note: Typically, the DBA is responsible for creating the database link. The dictionary view `USER_DB_LINKS` contains information on links to which a user has access.

Database Links

- **Create the database link.**

```
CREATE PUBLIC DATABASE LINK hq.acme.com
USING 'sales';
Database link created.
```

- **Write SQL statements that use the database link.**

```
SELECT *
FROM emp@HQ.ACME.COM;
```

ORACLE

Using Database Links

The example shown creates a database link. The `USING` clause identifies the service name of a remote database.

Once the database link is created, you can write SQL statements against the data in the remote site. If a synonym is set up, you can write SQL statements using the synonym.

For example:

```
CREATE PUBLIC SYNONYM HQ_EMP FOR emp@HQ.ACME.COM;
```

Then write a SQL statement that uses the synonym:

```
SELECT * FROM HQ_EMP;
```

You cannot grant privileges on remote objects.

Summary

In this lesson, you should have learned about DCL statements that control access to the database and database objects:

Statement	Action
`CREATE USER`	Creates a user (usually performed by a DBA)
`GRANT`	Gives other users privileges to access the your objects
`CREATE ROLE`	Creates a collection of privileges (usually performed by a DBA)
`ALTER USER`	Changes a user's password
`REVOKE`	Removes privileges on an object from users

ORACLE

Summary

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

- The DBA creates users who must have a password. The DBA is also responsible for establishing the initial system privileges for a user.
- Once the user has created an object, the user can pass along any of the available object privileges to other users or to all users by using the `GRANT` statement.
- A DBA can create roles by using the `CREATE ROLE` statement to pass along a collection of system or object privileges to multiple users. Roles make granting and revoking privileges easier to maintain.
- Users can change their password by using the `ALTER USER` statement.
- You can remove privileges from users by using the `REVOKE` statement.
- With data dictionary views, users can view the privileges granted to them and those that are granted on their objects.
- With database links, you can access data on remote databases. Privileges cannot be granted on remote objects.

Practice 13 Overview

This practice covers the following topics:

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

ORACLE

Practice 13 Overview

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

Practice 13

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

   _____

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

   _____

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

   _____

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

   _____

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

   _____

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

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		1700

8 rows selected.

8. Add a new row to your `DEPARTMENTS` table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.
9. Create a synonym for the other team's `DEPARTMENTS` table.

Practice 13 (continued)

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

Team 1 SELECT statement results:

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

9 rows selected.

Team 2 SELECT statement results:

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

9 rows selected.

Practice 13 (continued)

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

TABLE_NAME
COUNTRIES
DEPARTMENTS
EMPLOYEES
JOBS
JOB_GRADES
JOB_HISTORY
LOCATIONS
REGIONS

8 rows selected.

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

    Note: Your list may not exactly match the list shown below.

TABLE_NAME	OWNER
DEPARTMENTS	***owner***

13. Revoke the SELECT privilege on your table from the other team.

14

SQL Workshop

ORACLE

Workshop Overview

This workshop covers:

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

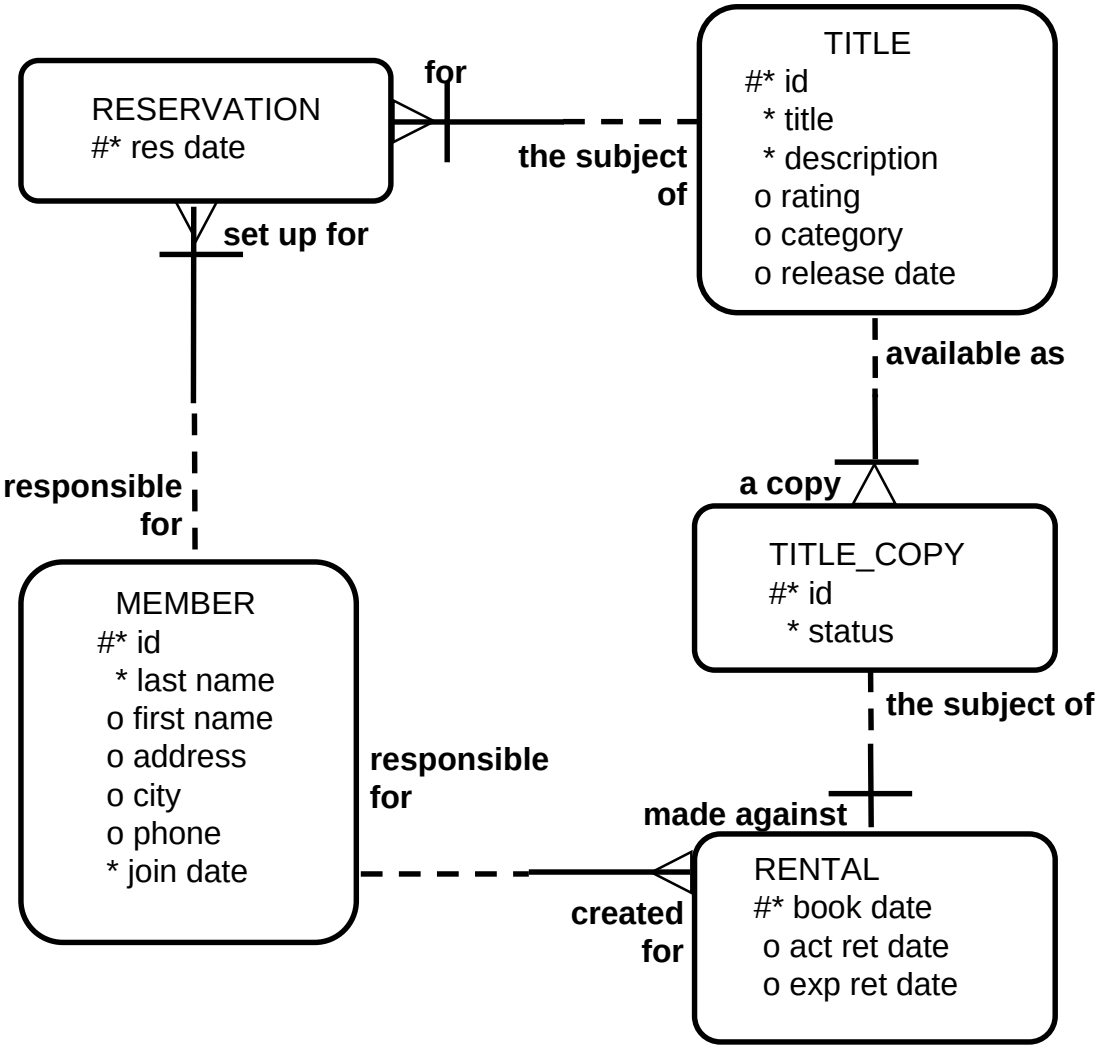
ORACLE

Workshop Overview

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

Note: If you want to build the tables, you can execute the commands in the `buildtab.sql` script in *i*SQL*Plus. If you want to drop the tables, you can execute the commands in `dropvid.sql` script in *i*SQL*Plus. Then you can execute the commands in `buildvid.sql` script in *i*SQL*Plus to create and populate the tables. If you use the `buildvid.sql` script to build and populate the tables, start with step 6b.

Video Application Entity Relationship Diagram



Practice 14

1. Create the tables based on the following table instance charts. Choose the appropriate data types and be sure to add integrity constraints.

   a. Table name: MEMBER

Column_ Name	MEMBER_ ID	LAST_ NAME	FIRST_NAM E	ADDRESS	CITY	PHONE	JOIN _ DATE
Key Type	PK						
Null/ Unique	NN,U	NN					NN
Default Value							System Date
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	25	25	100	30	15	

   b. Table name: TITLE

Column_ Name	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMEN TARY	
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	60	400	4	20	

Practice 14 (continued)

c. Table name: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/ Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Col		TITLE_ID	
Data Type	NUMBER	NUMBER	VARCHAR2
Length	10	10	15

d. Table name: RENTAL

Column Name	BOOK_ DATE	MEMBER_ ID	COPY_ ID	ACT_RET_ DATE	EXP_RET_ DATE	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Default Value	System Date				System Date + 2 days	
FK Ref Table		MEMBER	TITLE_ COPY			TITLE_ COPY
FK Ref Col		MEMBER_I D	COPY_ ID			TITLE_ID
Data Type	DATE	NUMBER	NUMBER	DATE	DATE	NUMBER
Length		10	10			10

Practice 14 (continued)

e. Table name: RESERVATION

Column Name	RES_ DATE	MEMBER_ ID	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		MEMBER_ID	TITLE_ID
Data Type	DATE	NUMBER	NUMBER
Length		10	10

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

TABLE_NAME
MEMBER
RENTAL
RESERVATION
TITLE
TITLE_COPY

CONSTRAINT_NAME	C	TABLE_NAME
MEMBER_LAST_NAME_NN	C	MEMBER
MEMBER_JOIN_DATE_NN	C	MEMBER
MEMBER_MEMBER_ID_PK	P	MEMBER
RENTAL_BOOK_DATE_COPY_TITLE_PK	P	RENTAL
RENTAL_MEMBER_ID_FK	R	RENTAL
RENTAL_COPY_ID_TITLE_ID_FK	R	RENTAL
RESERVATION_RESDATE_MEM_TIT_PK	P	RESERVATION
RESERVATION_MEMBER_ID	R	RESERVATION
TITLE_COPY_COPY_ID_TITLE_ID_PK	P	TITLE_COPY

18 rows selected.

Practice 14 (continued)

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

SEQUENCE_NAME
MEMBER_ID_SEQ
TITLE_ID_SEQ

4. Add data to the tables. Create a script for each set of data to add.
   a. Add movie titles to the `TITLE` table. Write a script to enter the movie information. Save the statements in a script named `lab14_4a.sql`. Use the sequences to uniquely identify each title. Enter the release dates in the `DD-MON-YYYY` format. Remember that single quotation marks in a character field must be specially handled. Verify your additions.

TITLE
Willie and Christmas Too
Alien Again
The Glob
My Day Off
Miracles on Ice
Soda Gang

6 rows selected.

Practice 14 (continued)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York.	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

Practice 14 (continued)

b. Add data to the `MEMBER` table. Place the insert statements in a script named `lab14_4b.sql`. Execute commands in the script. Be sure to use the sequence to add the member numbers.

First_ Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

Practice 14 (continued)

c. Add the following movie copies in the `TITLE_COPY` table:
Note: Have the `TITLE_ID` numbers available for this exercise.

Title	Copy_Id	Status
Willie and Christmas Too	1	AVAILABLE
Alien Again	1	AVAILABLE
	2	RENTED
The Glob	1	AVAILABLE
My Day Off	1	AVAILABLE
	2	AVAILABLE
	3	RENTED
Miracles on Ice	1	AVAILABLE
Soda Gang	1	AVAILABLE

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

Title_ Id	Copy_ Id	Member_ Id	Book_date	Exp_Ret_Date	Act_Ret_Date
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

Practice 14 (continued)

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

   Note: Your results may be different.

TITLE	COPY_ID	STATUS	EXP_RET_D
Alien Again	1	AVAILABLE	
Alien Again	2	RENTED	15-MAR-01
Miracles on Ice	1	AVAILABLE	
My Day Off	1	AVAILABLE	
My Day Off	2	AVAILABLE	
My Day Off	3	RENTED	16-MAR-01
Soda Gang	1	AVAILABLE	14-MAR-01
The Glob	1	AVAILABLE	
Willie and Christmas Too	1	AVAILABLE	15-MAR-01

9 rows selected.

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

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

Note: Your results may be different.

TITLE	COPY_ID	STATUS	EXP_RET_D
Alien Again	1	AVAILABLE	
Alien Again	2	RENTED	15-MAR-01
Interstellar Wars	1	RENTED	18-MAR-01
Interstellar Wars	2	AVAILABLE	
Miracles on Ice	1	AVAILABLE	
My Day Off	1	AVAILABLE	
My Day Off	2	AVAILABLE	
My Day Off	3	RENTED	16-MAR-01
Soda Gang	1	AVAILABLE	14-MAR-01
The Glob	1	AVAILABLE	
Willie and Christmas Too	1	AVAILABLE	15-MAR-01

11 rows selected.

7. Make a modification to one of the tables.

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

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

b. Create a script named `lab14_7b.sql` that contains update statements that update each video with a price according to the following list. Run the commands in the script.

Note: Have the `TITLE_ID` numbers available for this exercise.

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

c. Make sure that in the future all titles contain a price value. Verify the constraint.

CONSTRAINT_NAME	C	SEARCH_CONDITION
TITLE_PRICE_NN	C	"PRICE" IS NOT NULL

8. Create a report titled Customer History Report. This report contains each customer's history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the commands that generate the report in a script file named `lab14_8.sql`.

Note: Your results may be different.

Fri Mar 16 Customer History Report page 1

MEMBER	TITLE	BOOK_DATE	DURATION
Carmen Velasquez	Willie and Christmas Too	13-MAR-01	1
	Alien Again	15-MAR-01	
	Interstellar Wars	16-MAR-01	
LaDoris Ngao	My Day Off	14-MAR-01	
Molly Urguhart	Soda Gang	12-MAR-01	2

A

Practice Solutions

Practice 1 Solutions

1. Initiate an *i*SQL*Plus session using the user ID and password provided by the instructor.
2. *i*SQL*Plus commands access the database.
   False
3. The following SELECT statement executes successfully:
   True

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

4. The following SELECT statement executes successfully:
   True

```
SELECT *
FROM   job_grades;
```

5. There are four coding errors in this statement. Can you identify them?

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

   - **The EMPLOYEES table does not contain a column called sal. The column is called SALARY.**
   - **The multiplication operator is *, not x, as shown in line 2.**
   - **The ANNUAL SALARY alias cannot include spaces. The alias should read ANNUAL_SALARY or be enclosed in double quotation marks.**
   - **A comma is missing after the column, LAST_NAME.**

6. Show the structure of the DEPARTMENTS table. Select all data from the DEPARTMENTS table.

```
DESCRIBE departments

SELECT *
FROM   departments;
```

7. Show the structure of the EMPLOYEES table. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first. Provide an alias STARTDATE for the HIRE_DATE column. Save your SQL statement to a file named lab1_7.sql.

```
DESCRIBE employees

SELECT employee_id, last_name, job_id, hire_date StartDate
FROM   employees;
```

Practice 1 Solutions (continued)

8. Run your query in the file `lab1_7.sql`.

```
SELECT employee_id, last_name, job_id, hire_date
FROM   employees;
```

9. Create a query to display unique job codes from the `EMPLOYEES` table.

```
SELECT DISTINCT job_id
FROM   employees;
```

If you have time, complete the following exercises:

10. Copy the statement from `lab1_7.sql` into the *i*SQL*Plus Edit window. Name the column headings `Emp #`, `Employee`, `Job`, and `Hire Date`, respectively. Run your query again.

```
SELECT employee_id "Emp #", last_name "Employee",
       job_id "Job", hire_date "Hire Date"
FROM   employees;
```

11. Display the last name concatenated with the job ID, separated by a comma and space, and name the column `Employee and Title`.

```
SELECT last_name||', '||job_id "Employee and Title"
FROM   employees;
```

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

12. Create a query to display all the data from the `EMPLOYEES` table. Separate each column by a comma. Name the column `THE_OUTPUT`.

```
SELECT employee_id || ',' || first_name || ',' || last_name
      || ',' || email || ',' || phone_number || ','|| job_id
      || ',' || manager_id || ',' || hire_date || ',' ||
      salary || ',' || commission_pct || ',' || department_id
      THE_OUTPUT
FROM   employees;
```

Practice 2 Solutions

1. Create a query to display the last name and salary of employees earning more than $12,000. Place your SQL statement in a text file named `lab2_1.sql`. Run your query.

```
SELECT  last_name, salary
FROM    employees
WHERE   salary > 12000;
```

2. Create a query to display the employee last name and department number for employee number 176.

```
SELECT  last_name, department_id
FROM    employees
WHERE   employee_id = 176;
```

3. Modify `lab2_1.sql` to display the last name and salary for all employees whose salary is not in the range of $5,000 and $12,000. Place your SQL statement in a text file named `lab2_3.sql`.

```
SELECT  last_name, salary
FROM    employees
WHERE   salary NOT BETWEEN 5000 AND 12000;
```

4. Display the employee last name, job ID, and start date of employees hired between February 20, 1998, and May 1, 1998. Order the query in ascending order by start date.

```
SELECT   last_name, job_id, hire_date
FROM     employees
WHERE    hire_date BETWEEN '20-Feb-1998' AND '01-May-1998'
ORDER BY hire_date;
```

Practice 2 Solutions (continued)

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

```
SELECT   last_name, department_id
FROM     employees
WHERE    department_id IN (20, 50)
ORDER BY last_name;
```

6. Modify `lab2_3.sql` to list 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 `lab2_3.sql` as `lab2_6.sql`. Run the statement in `lab2_6.sql`.

```
SELECT   last_name "Employee", salary "Monthly Salary"
FROM     employees
WHERE    salary  BETWEEN 5000 AND 12000
AND      department_id IN (20, 50);
```

7. Display the last name and hire date of every employee who was hired in 1994.

```
SELECT   last_name, hire_date
FROM     employees
WHERE    hire_date LIKE '%94';
```

8. Display the last name and job title of all employees who do not have a manager.

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

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

```
SELECT   last_name, salary, commission_pct
FROM     employees
WHERE    commission_pct IS NOT NULL
ORDER BY salary DESC, commission_pct DESC;
```

Practice 2 Solutions (continued)

If you have time, complete the following exercises.

10. Display the last names of all employees where the third letter of the name is an *a*.

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

11. Display the last name of all employees who have an *a* and an *e* in their last name.

```
SELECT   last_name
FROM     employees
WHERE    last_name LIKE '%a%'
  AND    last_name LIKE '%e%';
```

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

12. Display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to $2,500, $3,500, or $7,000.

```
SELECT   last_name, job_id, salary
FROM     employees
WHERE    job_id IN ('SA_REP', 'ST_CLERK')
  AND    salary NOT IN (2500, 3500, 7000);
```

13. Modify `lab2_6.sql` to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave `lab2_6.sql` as `lab2_13.sql`. Rerun the statement in `lab2_13.sql`.

```
SELECT   last_name "Employee", salary "Monthly Salary",
         commission_pct
FROM     employees
WHERE    commission_pct = .20;
```

Practice 3 Solutions

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

```
SELECT   sysdate "Date"
FROM     dual;
```

2. For each employee, display the employee number, last_name, salary, and salary increased by 15% and expressed as a whole number. Label the column `New Salary`. Place your SQL statement in a text file named `lab3_2.sql`.

```
SELECT   employee_id, last_name, salary,
         ROUND(salary * 1.15, 0) "New Salary"
FROM     employees;
```

3. Run your query in the file `lab3_2.sql`.

```
SELECT  employee_id, last_name, salary,
        ROUND(salary * 1.15, 0) "New Salary"
FROM    employees;
```

4. Modify your query `lab3_2.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 `lab3_4.sql`. Run the revised query.

```
SELECT   employee_id, last_name, salary,
  ROUND(salary * 1.15, 0) "New Salary",
  ROUND(salary * 1.15, 0) - salary "Increase"
FROM     employees;
```

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

```
SELECT    INITCAP(last_name) "Name",
         LENGTH(last_name) "Length"
FROM     employees
WHERE    last_name LIKE 'J%'
OR       last_name LIKE 'M%'
OR       last_name LIKE 'A%'
ORDER BY last_name;
```

Practice 3 Solutions (continued)

6. For each employee, display the employee's last name, and calculate the number of months between today and the date the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

   Note: Your results will differ.

```
SELECT   last_name, ROUND(MONTHS_BETWEEN
               (SYSDATE, hire_date)) MONTHS_WORKED
FROM     employees
ORDER BY MONTHS_BETWEEN(SYSDATE, hire_date);
```

7. Write a query that produces the following for each employee: <employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.

```
SELECT   last_name || ' earns '
         || TO_CHAR(salary, 'fm$99,999.00')
         || ' monthly but wants '
         || TO_CHAR(salary * 3, 'fm$99,999.00')
         || '.' "Dream Salaries"
FROM     employees;
```

If you have time, complete the following exercises:

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

```
SELECT   last_name,
         LPAD(salary, 15, '$') SALARY
FROM     employees;
```

9. 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."

```
SELECT   last_name, hire_date,
  TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6),'MONDAY'),
         'fmDay, "the" Ddspth "of" Month, YYYY') REVIEW
FROM     employees;
```

10. 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.

```
SELECT   last_name, hire_date,
         TO_CHAR(hire_date, 'DAY') DAY
FROM     employees
ORDER BY TO_CHAR(hire_date - 1, 'd');
```

Practice 3 Solutions (continued)

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

11. Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, put "No Commission." Label the column COMM.

```
SELECT   last_name,
         NVL(TO_CHAR(commission_pct), 'No Commission') COMM
FROM     employees;
```

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

```
SELECT   rpad(last_name, 8)||' '|| rpad(' ', salary/1000+1, '*')
         EMPLOYEES_AND_THEIR_SALARIES
FROM     employees
ORDER BY salary DESC;
```

13. Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB_ID, as per 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

```
SELECT job_id, decode (job_id,
                       'ST_CLERK',  'E',
                       'SA_REP',    'D',
                       'IT_PROG',   'C',
                       'ST_MAN',    'B',
                       'AD_PRES',   'A',
                                    '0')GRADE
FROM employees;
```

Practice 3 Solutions (continued)

14. Rewrite the statement in the preceding question using the CASE syntax.

```
SELECT job_id, CASE job_id
       WHEN 'ST_CLERK' THEN 'E'
       WHEN 'SA_REP'   THEN 'D'
       WHEN 'IT_PROG'  THEN 'C'
       WHEN 'ST_MAN'   THEN 'B'
       WHEN 'AD_PRES'  THEN 'A'
       ELSE '0'  END  GRADE
FROM employees;
```

Practice 4 Solutions

1. Write a query to display the last name, department number, and department name for all employees.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

2. Create a unique listing of all jobs that are in department 80. Include the location of department in the output.

```
SELECT DISTINCT job_id, location_id
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND employees.department_id = 80;
```

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

```
SELECT e.last_name, d.department_name, d.location_id, l.city
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id
AND
d.location_id = l.location_id
AND e.commission_pct IS NOT NULL;
```

4. Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names. Place your SQL statement in a text file named `lab4_4.sql`.

```
SELECT last_name, department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND last_name LIKE '%a%';
```

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name, e.job_id, e.department_id,
      d.department_name
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id)
WHERE LOWER(l.city) = 'toronto';
```

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns `Employee`, `Emp#`, `Manager`, and `Mgr#`, respectively. Place your SQL statement in a text file named `lab4_6.sql`.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM employees w join employees m
ON (w.manager_id = m.employee_id);
```

Practice 4 Solutions (continued)

7. Modify `lab4_6.sql` to display all employees including King, who has no manager. Place your SQL statement in a text file named `lab4_7.sql`. Run the query in `lab4_7.sql`

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM employees w
LEFT OUTER JOIN employees m
ON (w.manager_id = m.employee_id);
```

If you have time, complete the following exercises.

8. Create a query 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.

```
SELECT e.department_id department, e.last_name employee,
       c.last_name colleague
FROM   employees e JOIN employees c
ON     (e.department_id = c.department_id)
WHERE   e.employee_id <> c.employee_id
ORDER BY e.department_id, e.last_name, c.last_name;
```

9. Show the structure of the `JOB_GRADES` table. Create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES
SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e, departments d, job_grades j
WHERE  e.department_id = d.department_id
AND    e.salary BETWEEN j.lowest_sal AND j.highest_sal;
-- OR
SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   job_grades j
ON    (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

Practice 4 Solutions (continued)

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

10. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT e.last_name, e.hire_date
FROM   employees e, employees davies
WHERE  davies.last_name = 'Davies'
AND    davies.hire_date < e.hire_date
-- OR
SELECT e.last_name, e.hire_date
FROM   employees e JOIN employees davies
ON     (davies.last_name = 'Davies')
WHERE  davies.hire_date < e.hire_date;
```

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns `Employee`, `Emp Hired`, `Manager`, and `Mgr Hired`, respectively.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w, employees m
WHERE  w.manager_id = m.employee_id
AND    w.hire_date <  m.hire_date;
-- OR
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w JOIN employees m
ON    (w.manager_id = m.employee_id)
WHERE    w.hire_date <  m.hire_date;
```

Practice 5 Solutions

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result.

   True

2. Group functions include nulls in calculations.

   False. Group functions ignore null values. If you want to include null values, use the NVL function.

3. The `WHERE` clause restricts rows prior to inclusion in a group calculation.

   True

4. Display the highest, lowest, sum, and average salary of all employees. Label the columns `Maximum`, `Minimum`, `Sum`, and `Average`, respectively. Round your results to the nearest whole number. Place your SQL statement in a text file named `lab5_6.sql`.

```
SELECT   ROUND(MAX(salary),0) "Maximum",
         ROUND(MIN(salary),0) "Minimum",
         ROUND(SUM(salary),0) "Sum",
         ROUND(AVG(salary),0) "Average"
FROM     employees;
```

5. Modify the query in `lab5_4.sql` to display the minimum, maximum, sum, and average salary for each job type. Resave `lab5_6.sql` to `lab5_4.sql`. Run the statement in `lab5_5.sql`.

```
SELECT   job_id, ROUND(MAX(salary),0) "Maximum",
                 ROUND(MIN(salary),0) "Minimum",
                 ROUND(SUM(salary),0) "Sum",
                 ROUND(AVG(salary),0) "Average"
FROM     employees
GROUP BY job_id;
```

Practice 5 Solutions (continued)

6. Write a query to display the number of people with the same job.

```
SELECT   job_id, COUNT(*)
FROM     employees
GROUP BY job_id;
```

7. Determine the number of managers without listing them. Label the column `Number of Managers`. *Hint: Use the `MANAGER_ID` column to determine the number of managers.*

```
SELECT   COUNT(DISTINCT manager_id) "Number of Managers"
FROM     employees;
```

8. Write a query that displays the difference between the highest and lowest salaries. Label the column `DIFFERENCE`.

```
SELECT   MAX(salary) - MIN(salary) DIFFERENCE
FROM     employees;
```

If you have time, complete the following exercises.

9. 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.

```
SELECT   manager_id, MIN(salary)
FROM     employees
WHERE    manager_id IS NOT NULL
GROUP BY manager_id
HAVING   MIN(salary) > 6000
ORDER BY MIN(salary) DESC;
```

10. Write a query to display each department's name, location, number of employees, and the average salary for all employees in that department. Label the columns `Name`, `Location`, `Number of People`, and `Salary`, respectively. Round the average salary to two decimal places.

```
SELECT   d.department_name "Name", d.location_id "Location",
         COUNT(*) "Number of People",
         ROUND(AVG(salary),2) "Salary"
FROM     employees e, departments d
WHERE    e.department_id = d.department_id
GROUP BY d.department_name, d.location_id;
```

Practice 5 Solutions (continued)

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

11. Create a query that will 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.

```
SELECT  COUNT(*) total,
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1995,1,0))"1995",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1996,1,0))"1996",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1997,1,0))"1997",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1998,1,0))"1998"
FROM    employees;
```

12. 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.

```
SELECT   job_id "Job",
         SUM(DECODE(department_id , 20, salary)) "Dept 20",
         SUM(DECODE(department_id , 50, salary)) "Dept 50",
         SUM(DECODE(department_id , 80, salary)) "Dept 80",
         SUM(DECODE(department_id , 90, salary)) "Dept 90",
         SUM(salary) "Total"
FROM     employees
GROUP BY job_id;
```

Practice 6 Solutions

1. Write a query to display the last name and hire date of any employee in the same department as Zlotkey. Exclude Zlotkey.

```
SELECT last_name, hire_date
FROM   employees
WHERE  department_id = (SELECT department_id
                        FROM   employees
                        WHERE  last_name = 'Zlotkey')
AND    last_nae <> 'Zlotkey';
```

2. Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in descending order of salary.

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary > (SELECT AVG(salary)
                 FROM   employees);
```

3. Write a query that displays the employee numbers and last names of all employees who work in a department with any employee whose last name contains a *u*. Place your SQL statement in a text file named `lab6_3.sql`. Run your query.

```
SELECT employee_id, last_name
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   employees
                         WHERE  last_name like '%u%');
```

4. Display the last name, department number, and job ID of all employees whose department location ID is 1700.

```
SELECT last_name, department_id, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   departments
                         WHERE  location_id = 1700);
```

Practice 6 Solutions (continued)

5. Display the last name and salary of every employee who reports to King.

```
SELECT last_name, salary
FROM   employees
WHERE  manager_id = (SELECT employee_id
                     FROM   employees
                     WHERE  last_name = 'King');
```

6. Display the department number, last name, and job ID for every employee in the Executive department.

```
SELECT department_id, last_name, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   departments
                         WHERE  department_name = 'Executive');
```

If you have time, complete the following exercises:

7. Modify the query in `lab6_3.sql` to display the employee numbers, last names, and salaries of all employees who earn more than the average salary and who work in a department with any employee with a *u* in their name. Resave `lab6_3.sql` to `lab6_7.sql`. Run the statement in `lab6_7.sql`.

```
SELECT employee_id, last_name, salary
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   employees
                         WHERE  last_name like '%u%')
AND    salary > (SELECT AVG(salary)
                 FROM   employees);
```

Practice 7 Solutions

Determine whether the following statements are true or false:

1. The following statement is correct:

   ```
   DEFINE & p_val = 100
   ```

   False

   The correct use of DEFINE is `DEFINE p_val=100`. The & is used within the SQL code.

2. The `DEFINE` command is a SQL command.

   False
   The `DEFINE` command is an iSQL*Plus command.

3. Write a script file to display the employee last name, job, and hire date for all employees who started between a given range. Concatenate the name and job together, separated by a space and comma, and label the column Employees. Use the `DEFINE` command to provide the two ranges. Use the format MM/DD/YYYY. Save the script file as `lab7_3.sql`.

```
SET ECHO OFF
SET VERIFY OFF
DEFINE low_date = 01/01/1998
DEFINE high_date = 01/01/1999
SELECT  last_name ||', '|| job_id EMPLOYEES, hire_date
FROM    employees
WHERE   hire_date BETWEEN TO_DATE('&low_date', 'MM/DD/YYYY')
                      AND TO_DATE('&high_date', 'MM/DD/YYYY')
/
UNDEFINE low_date
UNDEFINE high_date
SET VERIFY ON
SET ECHO ON
```

Practice 7 Solutions (continued)

4. Write a script to display the employee last name, job, and department name for a given location. The search condition should allow for case-insensitive searches of the department location. Save the script file as `lab7_4.sql`.

```
SET ECHO OFF
SET VERIFY OFF
COLUMN last_name HEADING "EMPLOYEE NAME"
COLUMN department_name HEADING "DEPARTMENT NAME”
SELECT  e.last_name, e.job_id, d.department_name
FROM    employees e, departments d, locations l
WHERE   e.department_id = d.department_id
AND     l.location_id = d.location_id
AND     l.city = INITCAP('&p_location')
/
COLUMN last_name CLEAR
COLUMN department_name CLEAR
SET VERIFY ON
SET ECHO ON
```

Practice 7 Solutions (continued)

5. Modify the code in `lab7_4.sql` to create a report containing the department name, employee last name, hire date, salary, and each employee's annual salary for all employees in a given location. Label the columns DEPARTMENT NAME, EMPLOYEE NAME, START DATE, SALARY, and ANNUAL SALARY, placing the labels on multiple lines. Resave the script as `lab7_5.sql` and execute the commands in the script.

```
SET ECHO OFF
SET FEEDBACK OFF
SET VERIFY OFF
BREAK ON department_name
COLUMN department_name HEADING "DEPARTMENT|NAME"
COLUMN last_name HEADING "EMPLOYEE|NAME"
COLUMN hire_date HEADING "START|DATE"
COLUMN salary HEADING "SALARY" FORMAT $99,990.00
COLUMN asal HEADING "ANNUAL|SALARY" FORMAT $99,990.00
SELECT d.department_name, e.last_name, e.hire_date,
       e.salary, e.salary*12 asal
FROM   departments d, employees e, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id   = l.location_id
AND    l.city          = '&p_location'
ORDER BY d.department_name
 /
COLUMN department_name CLEAR
COLUMN last_name CLEAR
COLUMN hire_date CLEAR
COLUMN salary CLEAR
COLUMN asal CLEAR
CLEAR BREAK
SET VERIFY ON
SET FEEDBACK ON
SET ECHO ON
```

Practice 8 Solutions

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab8_1.sql script to build the MY_EMPLOYEE table that will be used for the lab.

```
CREATE TABLE my_employee
(id  NUMBER(4) CONSTRAINT my_employee_id_nn NOT NULL,
 last_name VARCHAR2(25),
 first_name VARCHAR2(25),
 userid  VARCHAR2(8),
 salary  NUMBER(9,2));
```

2. Describe the structure of the MY_EMPLOYEE table to identify the column names.

```
DESCRIBE my_employee
```

3. Add the first row of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

```
INSERT INTO my_employee
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

4. Populate the MY_EMPLOYEE table with the second row of sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

```
INSERT INTO my_employee (id, last_name, first_name,
                         userid, salary)
VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5. Confirm your addition to the table.

```
SELECT   *
FROM     my_employee;
```

Practice 8 Solutions (continued)

6. Write an insert statement in a text file named `loademp.sql` to load rows into the `MY_EMPLOYEE` table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the userid.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        lower(substr('&p_first_name', 1, 1) ||
        substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
```

7. Populate the table with the next two rows of sample data by running the insert statement in the script that you created.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        lower(substr('&p_first_name', 1, 1) ||
        substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
```

8. Confirm your additions to the table.

```
SELECT   *
FROM my_employee;
```

9. Make the data additions permanent.

```
COMMIT;
```

Practice 8 Solutions (continued)

Update and delete data in the MY_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.

```
UPDATE  my_employee
SET     last_name = 'Drexler'
WHERE   id = 3;
```

11. Change the salary to 1000 for all employees with a salary less than 900.

```
UPDATE  my_employee
SET     salary = 1000
WHERE   salary < 900;
```

12. Verify your changes to the table.

```
SELECT  last_name, salary
FROM    my_employee;
```

13. Delete Betty Dancs from the MY_EMPLOYEE table.

```
DELETE
FROM  my_employee
WHERE last_name = 'Dancs';
```

14. Confirm your changes to the table.

```
SELECT  *
FROM    my_employee;
```

15. Commit all pending changes.

```
COMMIT;
```

Control data transaction to the MY_EMPLOYEE table.

16. Populate the table with the last row of sample data by modifying the statements in the script that you created in step 6. Run the statements in the script.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        lower(substr('&p_first_name', 1, 1) ||
        substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
```

Practice 8 Solutions (continued)

17. Confirm your addition to the table.

```
SELECT    *
FROM my_employee;
```

18. Mark an intermediate point in the processing of the transaction.

```
SAVEPOINT step_18;
```

19. Empty the entire table.

```
DELETE
FROM  my_employee;
```

20. Confirm that the table is empty.

```
SELECT *
FROM   my_employee;
```

21. Discard the most recent `DELETE` operation without discarding the earlier `INSERT` operation.

```
ROLLBACK TO step_18;
```

22. Confirm that the new row is still intact.

```
SELECT *
FROM   my_employee;
```

23. Make the data addition permanent.

```
COMMIT;
```

Practice 9 Solutions

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab9_1.sql, then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type		
Nulls/Unique		
FK Table		
FK Column		
Data type	Number	VARCHAR2
Length	7	25

```
CREATE TABLE dept
 (id NUMBER(7),
  name VARCHAR2(25));

DESCRIBE dept
```

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.

```
INSERT INTO dept
SELECT  department_id, department_name
FROM    departments;
```

3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab9_3.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				
FK Column				
Data type	Number	VARCHAR2	VARCHAR2	Number
Length	7	25	25	7

```
CREATE TABLE  emp
(id           NUMBER(7),
 last_name     VARCHAR2(25),
 first_name    VARCHAR2(25),
 dept_id       NUMBER(7));

DESCRIBE emp
```

4. Modify the EMP table to allow for longer employee last names. Confirm your modification.

```
ALTER TABLE emp
MODIFY (last_name   VARCHAR2(50));

DESCRIBE emp
```

5. Confirm that both the DEPT and EMP tables are stored in the data dictionary. (Hint: USER_TABLES)

```
SELECT   table_name
FROM     user_tables
WHERE    table_name IN ('DEPT', 'EMP');
```

6. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, and DEPARTMENT_ID columns. Name the columns in your new table ID, FIRST_NAME, LAST_NAME, SALARY , and DEPT_ID, respectively.

```
CREATE TABLE employees2 AS
SELECT  employee_id id, first_name, last_name, salary,
        department_id dept_id
FROM    employees;
```

7. Drop the EMP table.

```
DROP TABLE emp;
```

8. Rename the EMPLOYEES2 table to EMP.

```
RENAME employees2 TO emp;
```

Practice 9 Solutions (continued)

9. Add a comment to the DEPT and EMP table definitions describing the tables. Confirm your additions in the data dictionary.

```
COMMENT ON TABLE emp IS 'Employee Information';
COMMENT ON TABLE dept IS 'Department Information';
SELECT  *
FROM    user_tab_comments
WHERE   table_name = 'DEPT'
OR      table_name = 'EMP';
```

10. Drop the FIRST_NAME column from the EMP table. Confirm your modification by checking the description of the table.

```
ALTER TABLE emp
DROP COLUMN FIRST_NAME;

DESCRIBE emp
```

11. In the EMP table, mark the DEPT_ID column in the EMP table as UNUSED. Confirm your modification by checking the description of the table.

```
ALTER TABLE    emp
SET   UNUSED (dept_id);

DESCRIBE emp
```

12. Drop all the UNUSED columns from the EMP table. Confirm your modification by checking the description of the table.

```
ALTER TABLE emp
DROP UNUSED COLUMNS;

DESCRIBE emp
```

Practice 10 Solutions

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk

```
ALTER TABLE    emp
ADD CONSTRAINT my_emp_id_pk PRIMARY KEY (id);
```

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my_dept_id_pk.

```
ALTER TABLE    dept
ADD CONSTRAINT my_dept_id_pk PRIMARY KEY(id);
```

3. Add a column DEPT_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.

```
ALTER TABLE emp
ADD (dept_id NUMBER(7));

ALTER TABLE emp
ADD CONSTRAINT my_emp_dept_id_fk
FOREIGN KEY (dept_id) REFERENCES dept(id);
```

4. Confirm that the constraints were added by querying the USER_CONSTRAINTS view. Note the types and names of the constraints. Save your statement text in a file called lab10_4.sql.

```
SELECT   constraint_name, constraint_type
FROM     user_constraints
WHERE    table_name IN ('EMP', 'DEPT');
```

5. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP and DEPT tables. Notice that the new tables and a new index were created.

```
SELECT   object_name, object_type
FROM     user_objects
WHERE    object_name LIKE 'EMP%'
OR       object_name LIKE 'DEPT%';
```

If you have time, complete the following exercise:

6. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

```
ALTER TABLE EMP
ADD commission NUMBER(2,2)
CONSTRAINT my_emp_comm_ck CHECK (commission >= 0;
```

Practice 11 Solutions

1. Create a view called EMPLOYEES_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
CREATE OR REPLACE VIEW employees_vu AS
SELECT employee_id, last_name employee, department_id
FROM employees;
```

2. Display the contents of the EMPLOYEES_VU view.

```
SELECT   *
FROM     employees_vu;
```

3. Select the view name and text from the USER_VIEWS data dictionary view.

   Note: Another view already exists. The EMP_DETAILS_VIEW was created as part of your schema.

   Note: To see more contents of a LONG column, use the iSQL*Plus command SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

```
SET LONG 600
SELECT   view_name, text
FROM     user_views;
```

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

```
SELECT   employee, department_id
FROM     employees_vu;
```

5. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
CREATE VIEW dept50 AS
SELECT   employee_id empno, last_name employee,
         department_id deptno
FROM     employees
WHERE    department_id = 50
WITH CHECK OPTION CONSTRAINT emp_dept_50;
```

Practice 11 Solutions (continued)

6. Display the structure and contents of the DEPT50 view.

```
DESCRIBE dept50
SELECT   *
FROM     dept50;
```

7. Attempt to reassign Matos to department 80.

```
UPDATE   dept50
SET      deptno = 80
WHERE    employee = 'Matos';
```

If you have time, complete the following exercise:

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the EMPLOYEES, DEPARTMENTS, and JOB_GRADES tables. Label the columns Employee, Department, Salary, and Grade, respectively.

```
CREATE OR REPLACE VIEW salary_vu
AS
SELECT e.last_name "Employee",
       d.department_name "Department",
       e.salary "Salary",
       j.grade_level "Grades"
FROM   employees e,
       departments d,
       job_grades j
WHERE  e.department_id = d.department_id
AND    e.salary BETWEEN j.lowest_sal and j.highest_sal;
```

Practice 12 Solutions

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT_ID_SEQ.

```
CREATE SEQUENCE dept_id_seq
START WITH 200
INCREMENT BY 10
MAXVALUE 1000;
```

2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script lab12_2.sql. Run the statement in your script.

```
SELECT   sequence_name, max_value, increment_by, last_number
FROM     user_sequences;
```

3. Write a script to insert two rows into the DEPT table. Name your script lab12_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and Administration. Confirm your additions. Run the commands in your script.

```
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Education');

INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Administration');
```

4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

```
CREATE INDEX emp_dept_id_idx ON emp (dept_id);
```

5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table. Save the statement into a script named lab12_5.sql.

```
SELECT   index_name, table_name, uniqueness
FROM     user_indexes
WHERE    table_name = 'EMP';
```

Practice 13 Solutions

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

   The `CREATE SESSION` system privilege

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

   The `CREATE TABLE` privilege

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

   You can, or anyone you have given those privileges to by using the `WITH GRANT OPTION`.

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

   Create a role containing the system privileges and grant the role to the users

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

   The `ALTER USER` statement

6. Grant another user access to your `DEPARTMENTS` table. Have the user grant you query access to his or her `DEPARTMENTS` table.

   Team 2 executes the GRANT statement.

   ```
   GRANT select
   ON    departments
   TO    <user1>;
   ```

   Team 1 executes the GRANT statement.

   ```
   GRANT select
   ON    departments
   TO    <user2>;
   ```

   WHERE *user1* is the name of team 1 and *user2* is the name of team 2.

7. Query all the rows in your `DEPARTMENTS` table.

   ```
   SELECT  *
   FROM    departments;
   ```

8. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.

```
Team 1 executes this INSERT statement.
INSERT INTO departments(department_id, department_name)
VALUES (200, 'Education');
COMMIT;
Team 2 executes this INSERT statement.
INSERT INTO departments(department_id, department_name)
VALUES (210, 'Administration');
COMMIT;
```

9. Create a synonym for the other team's DEPARTMENTS table.

```
Team 1 creates a synonym named team2.
 CREATE SYNONYM    team2
 FOR <user2>.DEPARTMENTS;
Team 2 creates a synonym named team1.
 CREATE SYNONYM    team1
 FOR <user1>. DEPARTMENTS;
```

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

```
Team 1 executes this SELECT statement.
 SELECT  *
 FROM    team2;
Team 2 executes this SELECT statement.
 SELECT  *
 FROM    team1;
```

Practice 13 Solutions (continued)

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

```
SELECT  table_name
FROM    user_tables;
```

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

```
SELECT  table_name, owner
FROM    all_tables
WHERE   owner  <> <your account>;
```

13. Revoke the SELECT privilege from the other team.

Team 1 revokes the privilege.

```
REVOKE select
ON     departments
FROM   user2;
```

Team 2 revokes the privilege.

```
REVOKE select
ON     departments
FROM   user1;
```

Practice 14 Solutions

1. Create the tables based on the following table instance charts. Choose the appropriate data types and be sure to add integrity constraints.

   a. Table name: MEMBER

Column_ Name	MEMBER_ ID	LAST_ NAME	FIRST_NAM E	ADDRESS	CITY	PHONE	JOIN _ DATE
Key Type	PK						
Null/ Unique	NN,U	NN					NN
Default Value							System Date
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	25	25	100	30	15	

```
CREATE TABLE member
(member_id       NUMBER(10)
                 CONSTRAINT member_member_id_pk PRIMARY KEY,
last_name        VARCHAR2(25)
                 CONSTRAINT member_last_name_nn NOT NULL,
first_name       VARCHAR2(25),
address          VARCHAR2(100),
city VARCHAR2(30),
phone            VARCHAR2(15),
join_date        DATE DEFAULT SYSDATE
                 CONSTRAINT member_join_date_nn NOT NULL);
```

Practice 14 Solutions (continued)

b. Table name: TITLE

Column_ Name	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMENTARY	
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	60	400	4	20	

```
CREATE TABLE title
(title_id NUMBER(10)
    CONSTRAINT title_title_id_pk PRIMARY KEY,
 title         VARCHAR2(60)
    CONSTRAINT title_title_nn NOT NULL,
 description   VARCHAR2(400)
    CONSTRAINT title_description_nn NOT NULL,
 rating        VARCHAR2(4)
    CONSTRAINT title_rating_ck CHECK
         (rating IN ('G', 'PG', 'R', 'NC17', 'NR')),
 category VARCHAR2(20),
    CONSTRAINT title_category_ck CHECK
         (category IN ('DRAMA', 'COMEDY', 'ACTION',
         'CHILD', 'SCIFI', 'DOCUMENTARY')),
 release_date  DATE);
```

c. Table name: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/ Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Col		TITLE_ID	
Data Type	NUMBER	NUMBER	VARCHAR2
Length	10	10	15

```
CREATE TABLE title_copy
(copy_id         NUMBER(10),
 title_id        NUMBER(10)
 CONSTRAINT title_copy_title_if_fk REFERENCES title(title_id),
 status          VARCHAR2(15)
CONSTRAINT title_copy_status_nn NOT NULL
CONSTRAINT title_copy_status_ck CHECK (status IN
('AVAILABLE', 'DESTROYED','RENTED', 'RESERVED')),
CONSTRAINT title_copy_copy_id_title_id_pk
                PRIMARY KEY (copy_id, title_id));
```

d. Table name: RENTAL

Column Name	BOOK_DATE	MEMBER_ID	COPY_ID	ACT_RET_DATE	EXP_RET_DATE	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Default Value	System Date				System Date + 2 days	
FK Ref Table		MEMBER	TITLE_COPY			TITLE_COPY
FK Ref Col		MEMBER_ID	COPY_ID			TITLE_ID
Data Type	DATE	NUMBER	NUMBER	DATE	DATE	NUMBER
Length		10	10			10

```
CREATE TABLE rental
(book_date      DATE DEFAULT SYSDATE,
 member_id      NUMBER(10)
                CONSTRAINT rental_member_id_fk
                REFERENCES member(member_id),
 copy_id  NUMBER(10),
 act_ret_date DATE,
 exp_ret_date DATE DEFAULT SYSDATE + 2,
 title_id       NUMBER(10),
                CONSTRAINT rental_book_date_copy_title_pk
               PRIMARY KEY (book_date, member_id,
               copy_id,title_id),
                CONSTRAINT rental_copy_id_title_id_fk
               FOREIGN KEY (copy_id, title_id)
              REFERENCES title_copy(copy_id, title_id));
```

Practice 14 Solutions (continued)

e. Table name: RESERVATION

Column Name	RES_ DATE	MEMBER_ ID	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		MEMBER_ID	TITLE_ID
Data Type	DATE	NUMBER	NUMBER
Length		10	10

```
CREATE TABLE reservation
(res_date     DATE,
 member_id    NUMBER(10)
        CONSTRAINT reservation_member_id
        REFERENCES member(member_id),
 title_id     NUMBER(10)
        CONSTRAINT reservation_title_id
        REFERENCES title(title_id),
 CONSTRAINT reservation_resdate_mem_tit_pk PRIMARY KEY
        (res_date, member_id, title_id));
```

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

```
SELECT   table_name
FROM     user_tables
WHERE    table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
                        'RENTAL', 'RESERVATION');

SELECT   constraint_name, constraint_type, table_name
FROM     user_constraints
WHERE    table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
                        'RENTAL', 'RESERVATION');
```

3. Create sequences to uniquely identify each row in the `MEMBER` table and the `TITLE` table.

   a. Member number for the `MEMBER` table: start with 101; do not allow caching of the values. Name the sequence `MEMBER_ID_SEQ`.

```
CREATE SEQUENCE member_id_seq
START WITH 101
NOCACHE;
```

   b. Title number for the `TITLE` table: start with 92; no caching. Name the sequence `TITLE_ID_SEQ`.

```
CREATE SEQUENCE title_id_seq
START WITH 92
NOCACHE;
```

   c. Verify the existence of the sequences in the data dictionary.

```
SELECT  sequence_name, increment_by, last_number
FROM    user_sequences
WHERE   sequence_name IN ('MEMBER_ID_SEQ', 'TITLE_ID_SEQ');
```

4. Add data to the tables. Create a script for each set of data to add.
   a. Add movie titles to the `TITLE` table. Write a script to enter the movie information. Save the statements in a script named `lab14_4a.sql`. Use the sequences to uniquely identify each title. Enter the release dates in the `DD-MON-YYYY` format. Remember that single quotation marks in a character field must be specially handled. Verify your additions.

```
SET ECHO OFF
INSERT INTO title(title_id, title, description, rating,
       category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Willie and Christmas Too',
     'All of Willie''s friends make a Christmas list for
      Santa, but Willie has yet to add his own wish list.',
         'G', 'CHILD', TO_DATE('05-OCT-1995','DD-MON-YYYY')
/
INSERT INTO title(title_id , title, description, rating,
       category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Alien Again', 'Yet another
     installment of science fiction history.  Can the
     heroine save the planet from the alien life form?',
     'R', 'SCIFI', TO_DATE( '19-MAY-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
       category, release_date)
VALUES (title_id_seq.NEXTVAL, 'The Glob', 'A meteor crashes
     near a small American town and unleashes carnivorous
     goo in this classic.', 'NR', 'SCIFI',
       TO_DATE( '12-AUG-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
       category, release_date)
VALUES (title_id_seq.NEXTVAL, 'My Day Off', 'With a little
     luck and a lot ingenuity, a teenager skips school for
     a day in New York.', 'PG', 'COMEDY',
       TO_DATE( '12-JUL-1995','DD-MON-YYYY'))
/

...
COMMIT
/
SET ECHO ON

SELECT  title
FROM    title;
```

Practice 14 Solutions (continued)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York.	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

Practice 14 Solutions (continued)

b. Add data to the MEMBER table. Place the insert statements in a script named lab14_4b.sql. Execute commands in the script. Be sure to use the sequence to add the member numbers.

First_ Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO member(member_id, first_name, last_name, address,
            city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, '&first_name', '&last_name',
    '&address', '&city', '&phone', TO_DATE('&join_date',
   'DD-MM-YYYY');
COMMIT;
SET VERIFY ON
SET ECHO ON
```

c. Add the following movie copies in the `TITLE_COPY` table:
Note: Have the `TITLE_ID` numbers available for this exercise.

Title	Copy_Id	Status
Willie and Christmas Too	1	AVAILABLE
Alien Again	1	AVAILABLE
	2	RENTED
The Glob	1	AVAILABLE
My Day Off	1	AVAILABLE
	2	AVAILABLE
	3	RENTED
Miracles on Ice	1	AVAILABLE
Soda Gang	1	AVAILABLE

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 92, 'AVAILABLE');

INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 93, 'AVAILABLE');

INSERT INTO title_copy(copy_id, title_id, status)
VALUES (2, 93, 'RENTED');

INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 94, 'AVAILABLE');

INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 95, 'AVAILABLE');

INSERT INTO title_copy(copy_id, title_id,status)
VALUES (2, 95, 'AVAILABLE');

INSERT INTO title_copy(copy_id, title_id,status)
VALUES (3, 95, 'RENTED');

INSERT INTO title_copy(copy_id, title_id,status)
VALUES (1, 96, 'AVAILABLE');

INSERT INTO title_copy(copy_id, title_id,status)
VALUES (1, 97, 'AVAILABLE');
```

Practice 14 Solutions (continued)

d. Add the following rentals to the RENTAL table:

Note: Title number may be different depending on sequence number.

Title_ Id	Copy_ Id	Member_ Id	Book_date	Exp_Ret_Date	Act_Ret_Date
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

```
INSERT INTO rental(title_id, copy_id, member_id,
            book_date, exp_ret_date, act_ret_date)
VALUES (92, 1, 101, sysdate-3, sysdate-1, sysdate-2);
INSERT INTO rental(title_id, copy_id, member_id,
            book_date, exp_ret_date, act_ret_date)
VALUES (93, 2, 101, sysdate-1, sysdate-1, NULL);
INSERT INTO rental(title_id, copy_id, member_id,
            book_date, exp_ret_date, act_ret_date)
VALUES (95, 3, 102, sysdate-2, sysdate, NULL);
INSERT INTO rental(title_id, copy_id, member_id,
            book_date, exp_ret_date,act_ret_date)
VALUES (97, 1, 106, sysdate-4, sysdate-2, sysdate-2);
COMMIT;
```

Practice 14 Solutions (continued)

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

```
CREATE VIEW title_avail AS
SELECT   t.title, c.copy_id, c.status, r.exp_ret_date
FROM     title t, title_copy c, rental r
WHERE    t.title_id = c.title_id
AND      c.copy_id = r.copy_id(+)
AND      c.title_id = r.title_id(+);

SELECT   *
FROM     title_avail
ORDER BY title, copy_id;
```

6. Make changes to data in the tables.
   a. Add a new title. The movie is "Interstellar Wars," which is rated PG and classified as a science fiction movie. The release date is 07-JUL-77. The description is "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?" Be sure to add a title copy record for two copies.

```
INSERT INTO title(title_id, title, description, rating,
        category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Interstellar Wars',
        'Futuristic interstellar action movie.  Can the
        rebels save the humans from the evil Empire?',
        'PG', 'SCIFI', '07-JUL-77');
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (1, 98, 'AVAILABLE');
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (2, 98, 'AVAILABLE');
```

   b. Enter two reservations. One reservation is for Carmen Velasquez, who wants to rent "Interstellar Wars." The other is for Mark Quick-to-See, who wants to rent "Soda Gang."

```
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 101, 98);
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 104, 97);
```

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

```
INSERT INTO rental(title_id, copy_id, member_id)
VALUES (98,1,101);
UPDATE title_copy
SET    status= 'RENTED'
WHERE  title_id = 98
AND    copy_id = 1;
DELETE
FROM   reservation
WHERE  member_id = 101;
SELECT   *
FROM     title_avail
ORDER BY title, copy_id;
```

7. Make a modification to one of the tables.

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

```
ALTER TABLE title
ADD  (price NUMBER(8,2));
DESCRIBE title
```

Practice 14 Solutions (continued)

b. Create a script named `lab14_7b.sql` that contains update statements that update each video with a price according to the following list. Run the commands in the script.

Note: Have the `TITLE_ID` numbers available for this exercise.

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

```
SET ECHO OFF
SET VERIFY OFF
DEFINE price=
DEFINE title_id=
UPDATE title
SET    price = &price
WHERE  title_id = &title_id;
SET VERIFY OFF
SET ECHO OFF
```

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

```
ALTER TABLE title
MODIFY (price CONSTRAINT title_price_nn NOT NULL);
SELECT  constraint_name, constraint_type,
        search_condition
FROM    user_constraints
WHERE   table_name = 'TITLE';
```

Practice 14 Solutions (continued)

8. Create a report titled Customer History Report. This report contains each customer's history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the commands that generate the report in a script file named `lab14_8.sql`.

```
SET ECHO OFF
SET VERIFY OFF
TTITLE 'Customer History Report'
BREAK ON member  SKIP 1 ON REPORT
SELECT        m.first_name||' '||m.last_name MEMBER, t.title,
              r.book_date, r.act_ret_date - r.book_date DURATION
FROM          member m, title t, rental r
WHERE         r.member_id = m.member_id
AND           r.title_id = t.title_id
ORDER BY member;

CLEAR BREAK
TTITLE OFF
SET VERIFY ON
SET ECHO ON
```

B

Table Descriptions and Data

COUNTRIES Table

```
DESCRIBE countries
```

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

```
SELECT * FROM countries;
```

CO	COUNTRY_NAME	REGION_ID
CA	Canada	2
DE	Germany	1
UK	United Kingdom	1
US	United States of America	2

DEPARTMENTS Table

```
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)

```
SELECT * FROM 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		1700

8 rows selected.

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)

```
SELECT * FROM employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	

20 rows selected.

EMPLOYEES Table (continued)

	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
	24000			90
	17000		100	90
	17000		100	90
	9000		102	60
	6000		103	60
	4200		103	60
	5800		100	50
	3500		124	50
	3100		124	50
	2600		124	50
	2500		124	50
	10500	.2	100	80
	11000	.3	149	80
	8600	.2	149	80
	7000	.15	149	
	4400		101	10
	13000		100	20
	6000		201	20
	12000		101	110
T	8300		205	110

JOBS Table

```
DESCRIBE jobs
```

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

```
SELECT * FROM jobs;
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000

12 rows selected.

JOB_GRADES Table

```
DESCRIBE job_grades
```

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

```
SELECT * FROM job_grades;
```

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

6 rows selected.

JOB_HISTORY Table

```
DESCRIBE job_history
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

```
SELECT * FROM job_history;
```

EMPLOYEE_ID	START_DAT	END_DATE	JOB_ID	DEPARTMENT_ID
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

10 rows selected.

LOCATIONS Table

```
DESCRIBE locations
```

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

```
SELECT * FROM locations;
```

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

REGIONS Table

```
DESCRIBE regions
```

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

Using SQL*Plus

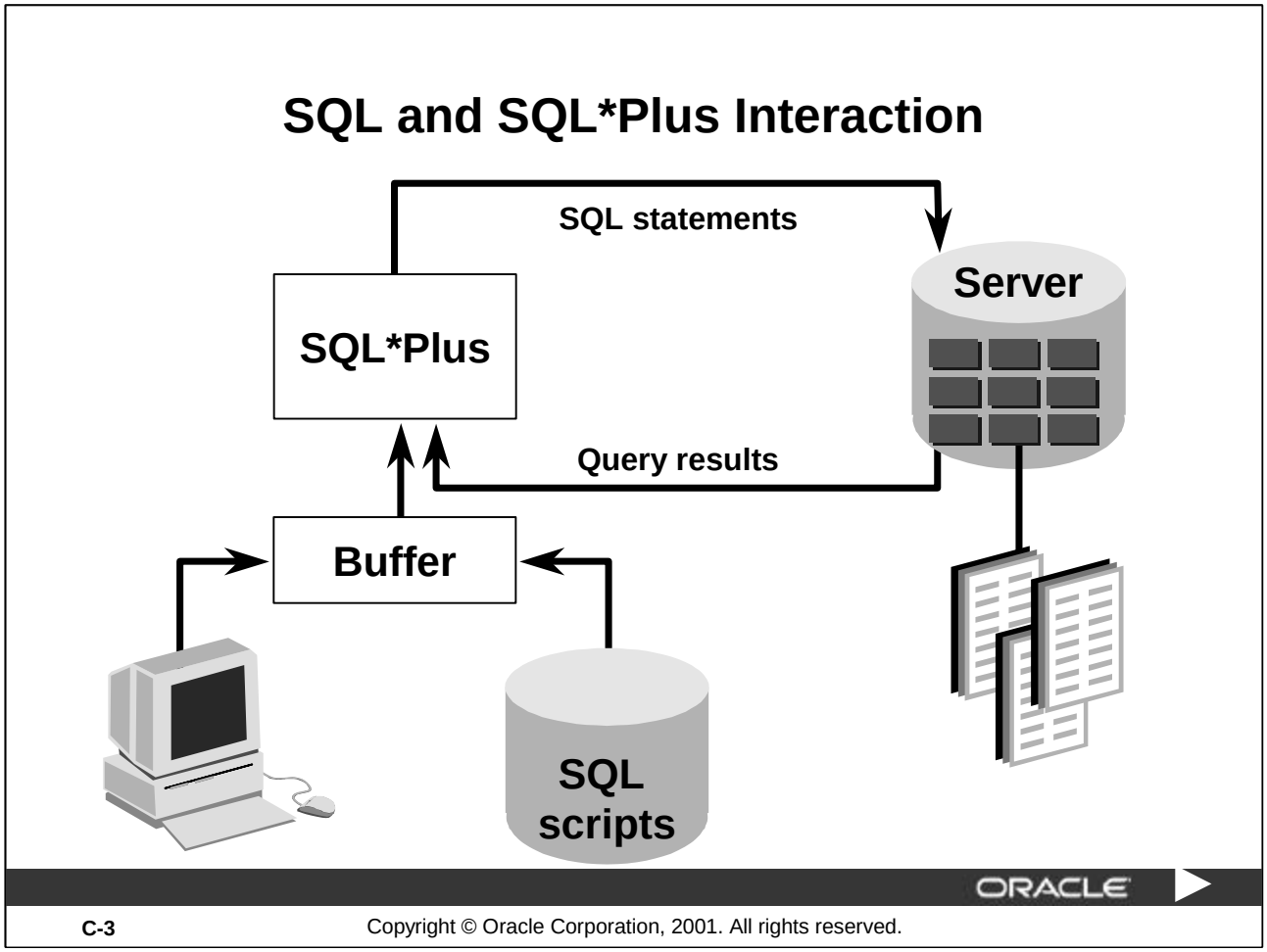
Objectives

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

- **Log in to SQL*Plus**
- **Edit SQL commands**
- **Format output using SQL*Plus commands**
- **Interact with script files**

Lesson Aim

You may want to create SELECT statements that can be used again and again. This lesson also covers the use of SQL*Plus commands to execute SQL statements. You learn how to format output using SQL*Plus commands, edit SQL commands, and save scripts in SQL*Plus.



SQL and SQL*Plus

SQL is a command language for communication with the Oracle9*i* Server from any tool or application. Oracle SQL contains many extensions. When you enter a SQL statement, it is stored in a part of memory called the *SQL buffer* and remains there until you enter a new SQL statement.

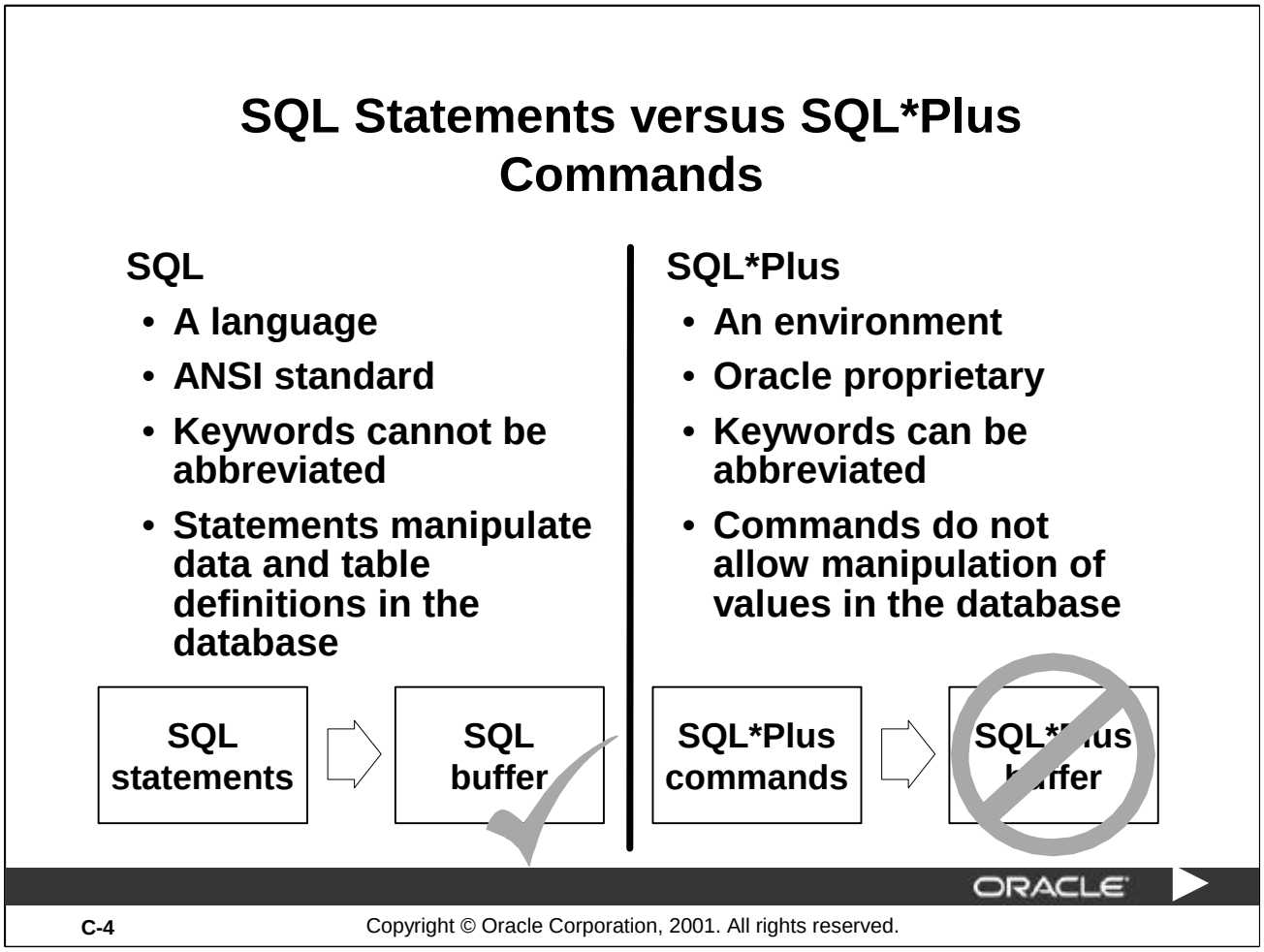
SQL*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle9*i* Server for execution. It contains its own command language.

Features of SQL

- SQL can be used by a range of users, including those with little or no programming experience.
- It is a nonprocedural language.
- It reduces the amount of time required for creating and maintaining systems.
- It is an English-like language.

Features of SQL*Plus

- SQL*Plus accepts ad hoc entry of statements.
- It accepts SQL input from files.
- It provides a line editor for modifying SQL statements.
- It controls environmental settings.
- It formats query results into basic reports.
- It accesses local and remote databases.



SQL and SQL*Plus (continued)

The following table compares SQL and SQL*Plus:

SQL	SQL*Plus
Is a language for communicating with the Oracle server to access data	Recognizes SQL statements and sends them to the server
Is based on American National Standards Institute (ANSI) standard SQL	Is the Oracle proprietary interface for executing SQL statements
Manipulates data and table definitions in the database	Does not allow manipulation of values in the database
Is entered into the SQL buffer on one or more lines	Is entered one line at a time, not stored in the SQL buffer
Does not have a continuation character	Uses a dash (-) as a continuation character if the command is longer than one line
Cannot be abbreviated	Can be abbreviated
Uses a termination character to execute commands immediately	Does not require termination characters; executes commands immediately
Uses functions to perform some formatting	Uses commands to format data

Overview of SQL*Plus

- **Log in to SQL*Plus.**
- **Describe the table structure.**
- **Edit your SQL statement.**
- **Execute SQL from SQL*Plus.**
- **Save SQL statements to files and append SQL statements to files.**
- **Execute saved files.**
- **Load commands from file to buffer to edit.**

SQL*Plus

SQL*Plus is an environment in which you can do the following:

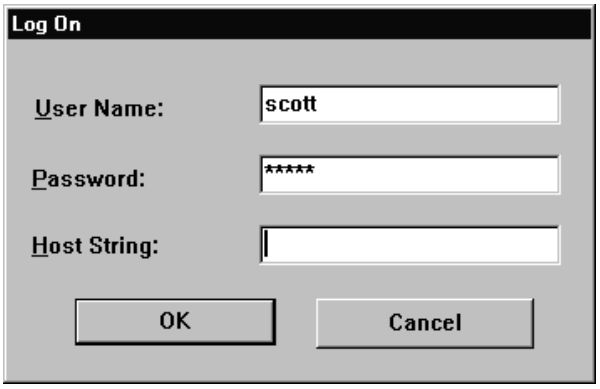
- Execute SQL statements to retrieve, modify, add, and remove data from the database
- Format, perform calculations on, store, and print query results in the form of reports
- Create script files to store SQL statements for repetitive use in the future

SQL*Plus commands can be divided into the following main categories:

Category	Purpose
Environment	Affect the general behavior of SQL statements for the session
Format	Format query results
File manipulation	Save, load, and run script files
Execution	Send SQL statements from SQL buffer to the Oracle server
Edit	Modify SQL statements in the buffer
Interaction	Create and pass variables to SQL statements, print variable values, and print messages to the screen
Miscellaneous	Connect to the database, manipulate the SQL*Plus environment, and display column definitions

Logging In to SQL*Plus

- **From a Windows environment:**



- **From a command line:**

```
sqlplus [username[/password
                [@database]]]
```


Logging In to SQL*Plus

How you invoke SQL*Plus depends on which type of operating system or Windows environment you are running.

To log in through a Windows environment:

1. Select Start > Programs > Oracle for Windows NT > SQL*Plus.
2. Fill in the username, password, and database name.

To log in through a command line environment:

1. Log on to your machine.
2. Enter the SQL*Plus command shown in the slide.

In the syntax:

username	your database username.
password	your database password (if you enter your password here, it is visible.)
@database	the database connect string.

Note: To ensure the integrity of your password, do not enter it at the operating system prompt. Instead, enter only your username. Enter your password at the Password prompt.

After you log in to SQL*Plus, you see the following message (if you are using SQL*Plus version 9*i*):

```
SQL*Plus: Release 9.0.1.0.0 - Development on Tue Jan 9 08:44:28 2001
(c) Copyright 2000 Oracle Corporation. All rights reserved.
```

Displaying Table Structure

Use the SQL*Plus `DESCRIBE` command to display the structure of a table.

```
DESC[RIBE] tablename
```

Displaying Table Structure

In SQL*Plus you can display the structure of a table using the `DESCRIBE` command. The result of the command is a display of column names and data types as well as an indication if a column must contain data.

In the syntax:

`tablename` the name of any existing table, view, or synonym that is accessible to the user

To describe the `JOB_GRADES` table, use this command:

```
SQL> DESCRIBE job_grades
Name                                      Null?    Type
----------------------------------------- -------- -----------
GRADE_LEVEL                                        VARCHAR2(3)
LOWEST_SAL                                         NUMBER
HIGHEST_SAL                                        NUMBER
```

Displaying Table Structure

```
SQL> 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)
```

Displaying Table Structure (continued)

The example in the slide displays the information about the structure of the DEPARTMENTS table.

In the result:

Null?	specifies whether a column *must* contain data; NOT NULL indicates that a column must contain data
Type	displays the data type for a column

The following table describes the data types:

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

SQL*Plus Editing Commands

- **A[PPEND] *text***
- **C[HANGE] / *old* / *new***
- **C[HANGE] / *text* /**
- **CL[EAR] BUFF[ER]**
- **DEL**
- **DEL *n***
- **DEL *m n***

SQL*Plus Editing Commands

SQL*Plus commands are entered one line at a time and are not stored in the SQL buffer.

Command	Description
A[PPEND] *text*	Adds text to the end of the current line
C[HANGE] / *old* / *new*	Changes *old* text to *new* in the current line
C[HANGE] / *text* /	Deletes *text* from the current line
CL[EAR] BUFF[ER]	Deletes all lines from the SQL buffer
DEL	Deletes current line
DEL *n*	Deletes line *n*
DEL *m n*	Deletes lines *m* to *n* inclusive

Guidelines

- If you press [Enter] before completing a command, SQL*Plus prompts you with a line number.
- You terminate the SQL buffer either by entering one of the terminator characters (semicolon or slash) or by pressing [Enter] twice. The SQL prompt then appears.

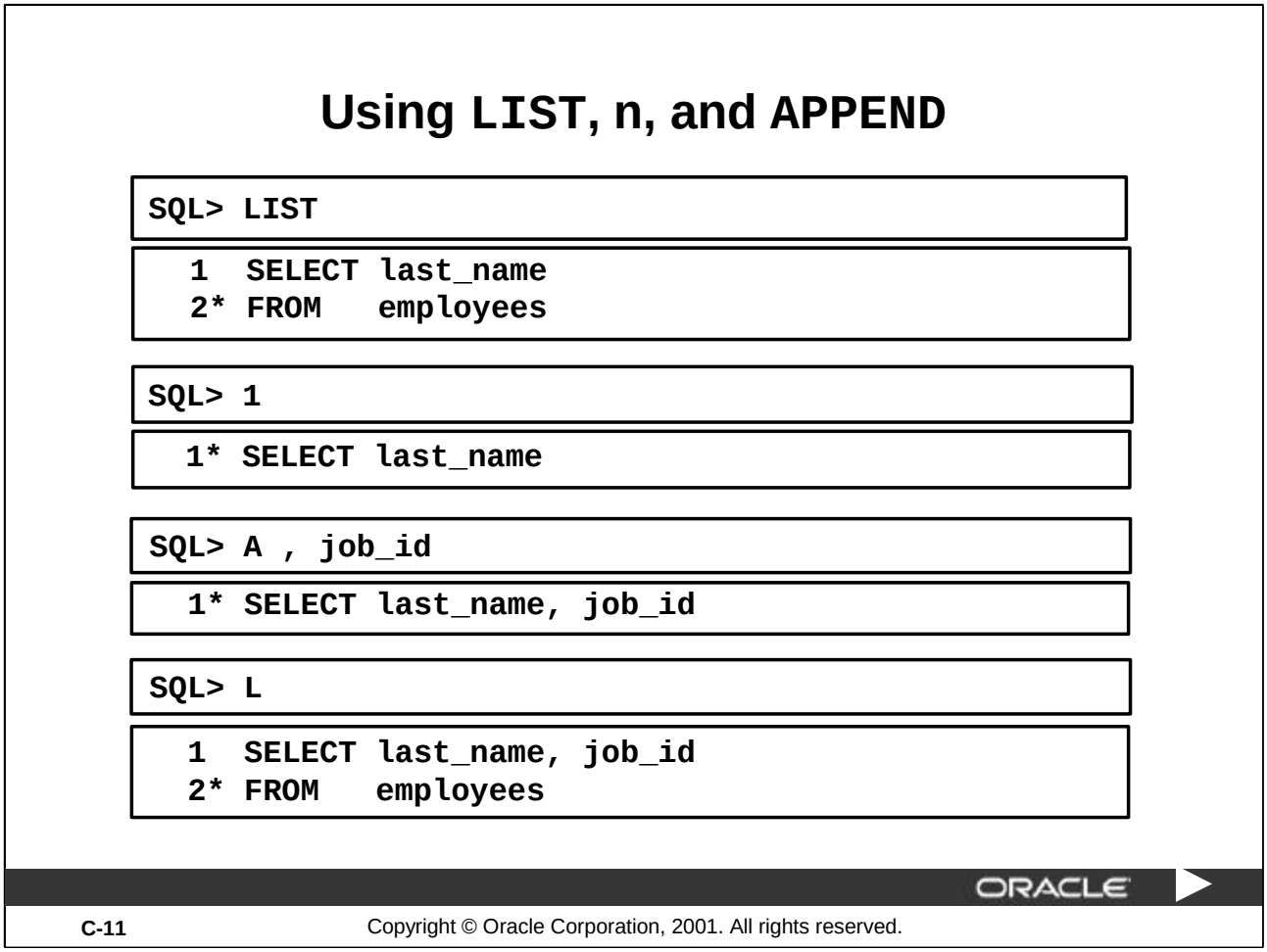
SQL*Plus Editing Commands

- **I[NPUT]**
- **I[NPUT]** ***text***
- **L[IST]**
- **L[IST]** ***n***
- **L[IST]** ***m n***
- **R[UN]**
- ***n***
- ***n text***
- **0** ***text***

SQL*Plus Editing Commands (continued)

Command	Description
I[NPUT]	Inserts an indefinite number of lines
I[NPUT] *text*	Inserts a line consisting of *text*
L[IST]	Lists all lines in the SQL buffer
L[IST] *n*	Lists one line (specified by *n*)
L[IST] *m n*	Lists a range of lines (*m* to *n)* inclusive
R[UN]	Displays and runs the current SQL statement in the buffer
n	Specifies the line to make the current line
n text	Replaces line *n* with *text*
0 *text*	Inserts a line before line 1

Note: You can enter only one SQL*Plus command per SQL prompt. SQL*Plus commands are not stored in the buffer. To continue a SQL*Plus command on the next line, end the first line with a hyphen (-).



Using LIST, n, and APPEND

- Use the L[IST] command to display the contents of the SQL buffer. The * beside line 2 in the buffer indicates that line 2 is the current line. Any edits that you made apply to the current line.
- Change the number of the current line by entering the number of the line you want to edit. The new current line is displayed.
- Use the A[PPEND] command to add text to the current line. The newly edited line is displayed. Verify the new contents of the buffer by using the LIST command.

Note: Many SQL*Plus commands including LIST and APPEND can be abbreviated to just their first letter. LIST can be abbreviated to L, APPEND can be abbreviated to A.

Using the CHANGE Command

```
SQL> L
  1* SELECT * from employees

SQL> c/employees/departments
  1* SELECT * from departments

SQL> L
  1* SELECT * from departments
```

Using the CHANGE Command

- Use L[IST] to display the contents of the buffer.
- Use the C[HANGE] command to alter the contents of the current line in the SQL buffer. In this case, replace the employees table with the departments table. The new current line is displayed.
- Use the L[IST] command to verify the new contents of the buffer.

SQL*Plus File Commands

- **SAVE *filename***
- **GET *filename***
- **START *filename***
- **@ *filename***
- **EDIT *filename***
- **SPOOL *filename***
- **EXIT**

SQL*Plus File Commands

SQL statements communicate with the Oracle server. SQL*Plus commands control the environment, format query results, and manage files. You can use the commands described in the following table:

Command	Description
SAV[E] *filename* [.ext] [REP[LACE]APP[END]]	Saves current contents of SQL buffer to a file. Use APPEND to add to an existing file; use REPLACE to overwrite an existing file. The default extension is .sql.
GET *filename* [.ext]	Writes the contents of a previously saved file to the SQL buffer. The default extension for the filename is .sql.
STA[RT] *filename* [.ext]	Runs a previously saved command file
@ *filename*	Runs a previously saved command file (same as START)
ED[IT]	Invokes the editor and saves the buffer contents to a file named afiedt.buf
ED[IT] [*filename*[.ext]]	Invokes the editor to edit contents of a saved file
SPO[OL] [*filename*[.ext]|OFF|OUT]	Stores query results in a file. OFF closes the spool file. OUT closes the spool file and sends the file results to the system printer.
EXIT	Leaves SQL*Plus

Using the SAVE and START Commands

```
SQL> L
  1  SELECT last_name, manager_id, department_id
  2* FROM   employees
SQL> SAVE my_query
```

```
Created file my_query
```

```
SQL> START my_query
```

```
LAST_NAME                 MANAGER_ID DEPARTMENT_ID
------------------------- ---------- -------------
King                                           90
Kochhar                          100           90
...
20 rows selected.
```

SAVE

Use the SAVE command to store the current contents of the buffer in a file. In this way, you can store frequently used scripts for use in the future.

START

Use the START command to run a script in SQL*Plus.

EDIT

Use the EDIT command to edit an existing script. This opens an editor with the script file in it. When you have made the changes, exit the editor to return to the SQL*Plus command line.

Summary

Use SQL*Plus as an environment to:

- **Execute SQL statements**
- **Edit SQL statements**
- **Format output**
- **Interact with script files**

Summary

SQL*Plus is an execution environment that you can use to send SQL commands to the database server and to edit and save SQL commands. You can execute commands from the SQL prompt or from a script file.

Index

Note: Bolded number refers to entire lesson.

A

Index

C

D

F

Index

M

N

R

S

S

T

U