GIFT School of Engineering and Applied Sciences

Spring 2020

CS-115/217: Database Systems

Lab-6 Manual

Using DDL Statements to Create and Manage Tables

Introduction to Lab

In this lab, you are introduced to the data definition language (DDL) statements. You are taught the basics of how to create simple tables, alter them, and remove them. The data types available in DDL are shown, and schema concepts are introduced. Constraints are tied into this lesson. Exception messages that are generated from violating constraints during DML are shown and explained.

The main topics of this lab include:

1. Database objects
2. Naming rules
3. The CREATE Table Statement
4. Referencing another user's Table
5. Creating tables
6. DEFAULT Options
7. Datatypes
8. Datetime Datatypes
9. Including Constraints
10. Constraint guidelines
11. Defining Constraints
12. NOT NULL Constraints
13. UNIQUE Constraints
14. PRIMARY KEY Constraints
15. FOREIGN KEY Constraints
16. FOREIGN KEY Constraints: Keywords
17. CREATE TABLE: Examples
18. Violating Constraints
19. The ALTER TABLE Statement
20. Adding a column
21. Dropping a column
22. Changing a column's Default Value
23. Changing a Column's datatype
24. Renaming Column
25. Renaming a Table
26. Guidelines for Altering Table
27. Dropping table
28. Truncating a table
29. Sample Exercises

Objectives of this Lab

At the end of this lab, students should be able to:

1. Categorize the main database objects
2. Review the table structure
3. List the data types that are available for columns
4. Create a simple table
5. Understand how constraints are created at the time of table creation
6. Describe how schema objects work

1. Database Objects

An Oracle database can contain multiple data structures. Each structure should be outlined in the database design so that it can be created during the build stage of database development.

1. **Table**: Store data
2. **View**: Subset of data from one or more tables
3. **Sequence**: Numeric value generator
4. **Index**: Improves the performance of some queries
5. **Synonym:** Gives alternative names to objects

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

Oracle Table Structures

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

2. Naming Rules

Name database tables and columns according to the standard rules for naming any Oracle database object:

- Table names and column names must begin with a letter and be 1–30 characters long.
- Names must contain only the characters A–Z, a–z, 0–9, _ (underscore), $, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Oracle Server user.
- Names must not be an Oracle Server reserved word.

Naming Guidelines

Use descriptive names for tables and other database objects.

Note: Names are case insensitive. For example, `EMPLOYEES` is treated as the same name as `eMPloyees` or `eMpLOYEES`.

3. The CREATE Table Statement

You create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the DDL statements, which are a subset of SQL statements used to create, modify, or remove Oracle database structures. These statements have an immediate effect on the database, and they also record information in the data dictionary.

To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator uses data control language statements to grant privileges to users.

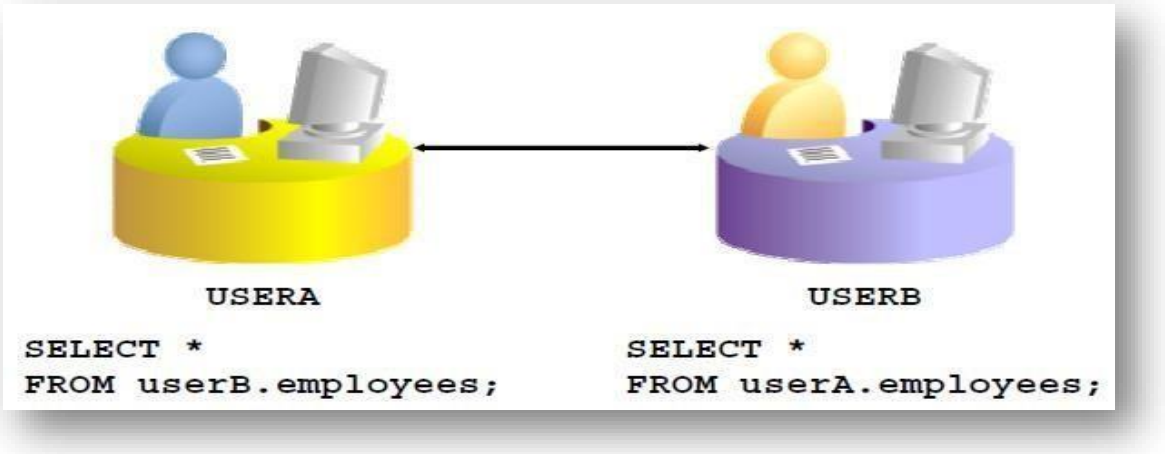
Syntax:

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

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

4. Referencing Another User's Tables

A *schema* is a collection of objects. Schema objects are the logical structures that directly refer to the data in a database. Schema objects include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.



If a table does not belong to the user, the owner's name must be prefixed to the table. For example, if there are schemas named **USERA** and **USERB,** and both have an **EMPLOYEES** table, then if **USERA** wants to access the **EMPLOYEES** table that belongs to **USERB**, he must prefix the table name with the schema name:

```
SELECT * FROM userb.employees;
```

If **USERB** wants to access the **EMPLOYEES** table that is owned by **USERA**, he must prefix the table name with the schema name:

```
SELECT * FROM usera.employees;
```

5. Creating Tables

```
CREATE TABLE dept_example (
      deptno NUMBER(2),
      dname VARCHAR2(14),
      loc VARCHAR2(13)

       );
```

Confirm the creation of the table:

```
DESC dept_example
```

The above example creates the DEPT_EXAMPLE table, with three columns: namely, DEPTNO, DNAME, and LOC. It further confirms the creation of the table by issuing the **DESC[RIBE]** command.

Note: Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

6. DEFAULT Options

When you define a table, you can specify that a column be given a default value by using the DEFAULT option. This option prevents null values from entering the columns if a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER), but the value cannot be the name of another column or a pseudo column (such as NEXTVAL or CURRVAL). The default expression must match the data type of the column.

Note: CURRVAL and NEXTVAL are explained later in this lab.

Example:

```
CREATE TABLE hire_dates (id
NUMBER(8),
hire_date DATE DEFAULT SYSDATE);

CREATE TABLE dept_example
(deptno NUMBER(2), dname
VARCHAR2(14), loc
VARCHAR2(13),
create_date DATE DEFAULT SYSDATE);

CREATE TABLE products (
     product_no NUMBER, name
     VARCHAR2(30), price
     NUMBER DEFAULT 9.99
);
```

Note: You may wish to test the default values by doing an INSERT into the above table, without giving explicit values to the DEFAULT columns.

7. Data Types

When you identify a column for a table, you need to provide a data type for the column.

There are several data types available:

Data Type	Description
VARCHAR2(*size*)	Variable-length character data (A maximum *size* must be specified: minimum *size* is 1; maximum *size* is 4,000.)
CHAR [(*size*)]	Fixed-length character data of length *size* bytes (Default and minimum *size* is 1; maximum *size* is 2,000.)

8. Datetime Data Types

Data Type	**Description**
TIMESTAMP	Enables the time to be stored as a date with fractional seconds. There are several variations of this data type.
INTERVAL YEAR TO MONTH	Enables time to be stored as an interval of years and months. Used to represent the difference between two datetime values in which the only significant portions are the year and month.
INTERVAL DAY TO SECOND	Enables time to be stored as an interval of days, hours, minutes, and seconds. Used to represent the precise difference between two datetime values.

9. Including Constraints

The Oracle server uses constraints to prevent invalid data entry into tables.

You can use constraints to do the following:

- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables

Data Integrity Constraints

Constraint	**Description**

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

10. Constraint Guidelines

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

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

11. Defining Constraints

Syntax:

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

Column-level constraint:

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

Table-level constraint:

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

(column, ...),
```

You can create the constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition and must refer to the column or columns on which the constraint pertains in a set of parentheses.

- **NOT NULL** constraints must be defined at the column level.

Constraints that apply to more than one column must be defined at the table level.

In the syntax:

- **schema** is the same as the owner's name
- **table** is the name of the table
- **DEFAULT expr** specifies a default value to use if a value is omitted in the **INSERT** statement
- **column** is the name of the column
- **datatype** is the column's data type and length
- **column_constraint** is an integrity constraint as part of the column definition
- **table_constraint** is an integrity constraint as part of the table definition

Examples:

Column-level constraint:

```
CREATE TABLE employees( employee_id
NUMBER(6)
CONSTRAINT emp_emp_id_pk PRIMARY KEY, first_name
VARCHAR2(20),
...);
```

Table-level constraint:

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

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

Both examples create a primary key constraint on the **EMPLOYEE_ID** column of the **EMPLOYEES** table.

1. The first example uses the column-level syntax to define the constraint.
2. The second example uses the table-level syntax to define the constraint. More details about the primary key constraint are provided later in this lab.

12. NOT NULL Constraint

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default. NOT NULL constraints must be defined at the column level.

```
CREATE TABLE copy_employees(
employee_id NUMBER(6), last_name
VARCHAR2(25) NOT NULL,
. . .
```

13. UNIQUE Constraint

A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or set of columns. The column (or set of columns) included in the definition of the UNIQUE key constraint is called the *unique key*. If the UNIQUE constraint comprises more than one column, that group of columns is called a *composite unique key*.
UNIQUE constraints enable the input of nulls unless you also define NOT NULL constraints for the same columns. In fact, any number of rows can include nulls for columns without NOT NULL constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite UNIQUE key) always satisfies a UNIQUE constraint.

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

```
CREATE TABLE copy_employees(
employee_id NUMBER(6), last_name
VARCHAR2(25) NOT NULL, email
VARCHAR2(25), salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE
NOT NULL,
...
CONSTRAINT emp_email_uk UNIQUE(email));
```

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

The above example applies the UNIQUE constraint to the EMAIL column of the COPY_EMPLOYEES table. The name of the constraint is EMP_EMAIL_UK.

Note: The Oracle server enforces the UNIQUE constraint by implicitly creating a unique index on the unique key column or columns.

14. PRIMARY KEY Constraint

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

Note: Because uniqueness is part of the primary key constraint definition, the Oracle server enforces the uniqueness by implicitly creating a unique index on the primary key column or columns.

```
CREATE TABLE copy_employees
( employee_id NUMBER(6)
CONSTRAINT emp_employee_id PRIMARY KEY
, first_name VARCHAR2(20)
, last_name VARCHAR2(25) . . .
```

Table-Level:

```
CREATE TABLE users (
  UserID INTEGER NOT NULL,
  First_Name varchar(20),
  Last_Name varchar(20),
  ZipCode varchar(10),
  PRIMARY KEY(UserID));
```

15. FOREIGN KEY Constraint

The FOREIGN KEY (or referential integrity) constraint designates a column or combination of columns as a foreign key and establishes a relationship between a primary key or a unique key in the same table or a different table.

Guidelines

- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, rather than physical, pointers.

```
CREATE TABLE copy_employees(
employee_id NUMBER(6), last_name
VARCHAR2(25) NOT NULL, email
VARCHAR2(25), salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE NOT NULL,
... department_id
NUMBER(4),
CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
REFERENCES departments(department_id),
CONSTRAINT emp_email_uk UNIQUE(email));
```

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

The above example defines a **FOREIGN KEY** constraint on the **DEPARTMENT_ID** column of the **EMPLOYEES** table, using table-level syntax. The name of the constraint is

EMP_DEPTID_FK.

The foreign key can also be defined at the column level, provided the constraint is based on a single column. The syntax differs in that the keywords **FOREIGN KEY** do not appear. For example:

```
CREATE TABLE copy_employees
(... department_id NUMBER(4) CONSTRAINT
emp_deptid_fk REFERENCES
departments(department_id), ... )
```

16. FOREIGN KEY Constraint: Keywords

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

- **FOREIGN KEY** is used to define the column in the child table at the table-constraint level.
- **REFERENCES** identify the table and column in the parent table.
- **ON DELETE CASCADE** indicates that when the row in the parent table is deleted, the dependent rows in the child table are also deleted.
- **ON DELETE SET NULL** converts foreign key values to null when the parent value is removed.

The default behavior is called the *restrict rule*, which disallows the update or deletion of referenced data.

Without the **ON DELETE CASCADE** or the **ON DELETE SET NULL** options, the row in the parent table cannot be deleted if it is referenced in the child table.

17. CREATE TABLE: Example

```
CREATE TABLE employees
( employee_id NUMBER(6)
CONSTRAINT emp_employee_id PRIMARY KEY
, first_name VARCHAR2(20)
, last_name VARCHAR2(25)
CONSTRAINT emp_last_name_nn NOT NULL
, email VARCHAR2(25)
CONSTRAINT emp_email_nn NOT NULL
CONSTRAINT emp_email_uk UNIQUE
, phone_number VARCHAR2(20)
, hire_date DATE
CONSTRAINT emp_hire_date_nn NOT NULL
, job_id VARCHAR2(10)
CONSTRAINT emp_job_nn NOT NULL
, salary NUMBER(8,2)
CONSTRAINT emp_salary_ck CHECK (salary>0)
, commission_pct NUMBER(2,2)
, manager_id NUMBER(6)
, department_id NUMBER(4)
CONSTRAINT emp_dept_fk REFERENCES
departments (department_id));
```

18. Violating Constraints

When you have constraints in place on columns, an error is returned to you if you try to violate the constraint rule.
For example, if you attempt to update a record with a value that is tied to an integrity constraint, an error is returned.

```
UPDATE employees
SET department_id = 55
WHERE department_id = 110;
```

In this example, department 55 does not exist in the parent table, DEPARTMENTS, and so you receive the *parent key* violation ORA-02291.

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

```
DELETE FROM departments
WHERE department_id = 60;
```

The example in the slide tries to delete department 60 from the DEPARTMENTS table, but it results in an error because that department number is used as a foreign key in the EMPLOYEES table. If the parent record that you attempt to delete has child records, then you receive the *child record found* violation ORA-02292.

The following statement works because there are no employees in department 70:

```
DELETE FROM departments
WHERE department_id = 70;
```

19. The ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

Syntax:

```
1. ALTER TABLE table
   ADD COLUMN column datatype
      [, COLUMN column datatype]...;

2. ALTER TABLE table DROP COLUMN column;

3. ALTER TABLE table
   ALTER COLUMN column TYPE datatype
      [, COLUMN column TYPE datatype]...;

4. ALTER TABLE table
   ALTER COLUMN column SET DEFAULT expression
```

In the syntax:

table	is the name of the table

ADD|ALTER|DROP	is the type of modification
column	is the name of the new column
datatype	is the data type and length of the new column
DEFAULT *expr*	specifies the default value for a new column

20. Adding a Column

Use the ADD clause to add columns.

Example,

```
1. ALTER TABLE dept30
      ADD COLUMN job VARCHAR(9);

2. ALTER TABLE products
      ADD COLUMN description text;
```

Note: The new column becomes the last column.

Guidelines for Adding a Column

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

The above example adds a column named JOB to the DEPT30 table. The JOB column becomes the last column in the table.

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

You can also define constraints on the column at the same time, using the usual syntax:

```
ALTER TABLE products
    ADD COLUMN description text CHECK (description <> '');
```

Working with table constraints will be presented in length in the next lab.

21. Dropping a Column

To remove a column, use a command like:

```
ALTER TABLE products DROP
      COLUMN description;
```

Whatever data was in the column disappears. Table constraints involving the column are dropped, too. However, if the column is referenced by a foreign key constraint of another table,

PostgreSQL will not silently drop that constraint. You can authorize dropping everything that depends on the column by adding CASCADE:

```
ALTER TABLE products
    DROP COLUMN description CASCADE;
```

Please consult the Lab-11 manual for a description of applying keys to columns and the CASCADE option.

22. Changing a Column's Default Value

To set a new default for a column, use a command like:

```
ALTER TABLE products
     ALTER COLUMN price SET DEFAULT 7.77;
```

Note that this doesn't affect any existing rows in the table; it just changes the default for future INSERT commands.

To remove any default value, use:

```
ALTER TABLE products
    ALTER COLUMN price DROP DEFAULT;
```

This is effectively the same as setting the default to null. As a consequence, it is not an error to drop a default where one hadn't been defined, because the default is implicitly the null value.

23. Changing a Column's Data Type

To convert a column to a different data type, use a command like:

```
ALTER TABLE products
    ALTER COLUMN price TYPE numeric(10,2);
```

This will succeed only if each existing entry in the column can be converted to the new type by an implicit cast. If a more complex conversion is needed, you can add a USING clause that specifies how to compute the new values from the old.

Note: PostgreSQL will attempt to convert the column's default value (if any) to the new type, as well as any constraints that involve the column. But these conversions might fail, or might produce surprising results. It's often best to drop any constraints on the column before altering its type, and then add back suitably modified constraints afterwards.

24. Renaming a Column

To rename a column:

```
ALTER TABLE products
    RENAME COLUMN product_no TO product_number;
```

25. Renaming a Table

To rename a table:

```
ALTER TABLE products
    RENAME TO items;
```

26. Guidelines for Altering Tables

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

27. Dropping a Table

The **DROP TABLE** statement removes the definition of a PostgreSQL table. When you drop a table, the database loses all the data in the table and all the indexes associated with it.

Syntax

```
DROP TABLE tablename
```

In the syntax:

tablename is the name of the table

Example,

```
DROP TABLE dept30;
```

To destroy two tables, `films` and `distributors`:

```
DROP TABLE films, distributors;
```

Guidelines

- All data is deleted from the table.
- Any views and synonyms remain but are invalid.
- Any pending transactions are committed.
- Only the creator of the table can remove a table.

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

28. Truncating a Table

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

Syntax

```
TRUNCATE [TABLE] table;
```

In the syntax:

table is the name of the table

Example,

```
TRUNCATE TABLE dept30;
```

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

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

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

29. Sample Exercises

1. Create the TEST_DEPT table based on the following table instance chart.

Column Name	ID	NAME
Data type	NUMERIC	VARCHAR
Length	7	25

Confirm that the table is created.

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

3. Create the TEST_EMP table based on the following table instance chart.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Data type	NUMERIC	VARCHAR	VARCHAR	NUMERIC
Length	7	25	25	7

Confirm that the table is created.

4. Populate the TEST_EMP table with data from the EMP table. Include only columns that you need.

5. Modify the TEST_EMP table to allow for longer employee last names (length: 50). Confirm your modification.

6. Create the EMPLOYEES2 table based on the structure of the EMP table. Include only the EMPNO, ENAME, SAL, and DEPTNO columns. Name the columns in your new table ID, EMP_NAME, SALARY, and DEPT_ID, respectively.

7. Create a complete copy of the TEST_EMP table. Drop the TEST_EMP table.

8. Rename the EMPLOYEES2 table as MY_EMP.

9. Drop the FIRST_NAME column from the TEST_EMP table. Confirm your modification by checking the description of the table.

10. Give yourself enough practice of altering tables by using all the options of the ALTER TABLE statement on the TEST_EMP, and TEST_DEPT tables.