DP

12-1 INSERT Statements

Objectives

In this lesson, you will learn to:

• Explain the importance of being able to alter the data in a database
• Construct and execute INSERT statements which insert a single row using a VALUES clause
• Construct and execute INSERT statements that use special values, null values, and date values
• Construct and execute INSERT statements that copy rows from one table to another using a subquery

data manipulation language (DML)

```
CREATE TABLE copy_tablename
AS (SELECT * FROM tablename);
```

INSERT

• The INSERT statement is used to add a new row to a table. The statement requires three values:
– the name of the table
– the names of the columns in the table to populate
– corresponding values for each column
• How can we INSERT the data below to create a new department in the copy_departments table?

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
200	Human Resources	205	1500

• The syntax below uses INSERT to add a new department to the copy_departments table.
• This statement explicitly lists each column as it appears in the table.
• The values for each column are listed in the same order.
– Note that number values are not enclosed in single quotation marks.

```
INSERT INTO copy_departments (department_id, department_name, manager_id, location_id)
VALUES (200,'Human Resources', 205, 1500);
```

• Another way to insert values in a table is to implicitly add them by omitting the column names.
• One precaution: the values for each column must match exactly the default order in which they appear in the table (as shown in a DESCRIBE statement), and a value must be provided for each column.
• The INSERT statement in this example was written without explicitly naming the columns.
• For clarity, however, it is best to use the column names in an INSERT clause.

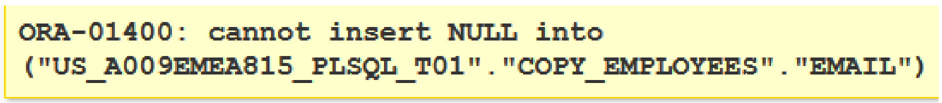
```
INSERT INTO copy_departments
VALUES (210,'Estate Management', 102, 1700);
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
210	Estate Management	102	1700

Table Summary

• As shown in the example, the table summary provides information about each column in the table, such as:
– the allowance of duplicate values
– the type of data allowed
– the amount of data allowed
– the allowance of NULL values
• Notice where the “Data Type” column is a character data type the “Length” column specifies the maximum number of characters permitted.
• For Number data types the brackets specify the Precision and Scale.
• Precision is the total number of digits, and Scale is the number of digits to the right of the decimal place.
• The INSERT statement need not specify every column—the Nullable columns may be excluded.
• If every column that requires a value is assigned a value, the insert works.
• In our example, the EMAIL column is defined as a NOT NULL column.
• An implicit attempt to add values to the table as shown would generate an error.

```
INSERT INTO copy_employees (employee_id, first_name, last_name, phone_number, hire_date, job_id, salary)
VALUES (302,'Grigorz','Polanski', '8586667641', '15-Jun-2015','IT_PROG',4200);
```



```
ORA-01400: cannot insert NULL into
("US_A009EMEA815_PLSQL_T01"."COPY_EMPLOYEES"."EMAIL")
```

• An implicit insert will automatically insert a null value in columns that allow nulls.
• To explicitly add a null value to a column that allows nulls, use the NULL keyword in the VALUES list.
• To specify empty strings and/or missing dates, use empty single quotation marks (with no spaces between them like this '') for the missing data.

```
INSERT INTO copy_employees
        (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary)
VALUES (302,'Grigorz','Polanski', 'gpolanski', '', '15-Jun-2015', 'IT_PROG',4200);
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMM_PCT	MGR_ID	DEPT_ID	BONUS
302	Grigorz	Polanski	gpolanski	-	15-Jun-2015	IT_PROG	4200	-	-	-	-

Inserting Special Values
• Special values such as SYSDATE and USER can be entered in the VALUES list of an INSERT statement.
• SYSDATE will put the current date and time in a column.
• USER will insert the current session's username, which is OAE_PUBLIC_USER in Oracle Application Express.
• This example adds USER as the last name, and SYSDATE for hire date.

```
INSERT INTO copy_employees
        (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary)
VALUES (304,'Test',USER, 't_user', 4159982010, SYSDATE, 'ST_CLERK',2500);
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMM_PCT	MGR_ID	DEPT_ID	BONUS
304	Test	APEX_PUBLIC_USER	t_user	4159982010	15-Jun-2015	ST_CLERK	2500	-	-	-	-

Inserting Specific Date Values
• The default format model for date data types is DD-Mon-YYYY.
• With this date format, the default time of midnight (00:00:00) is also included.
• In an earlier section, we learned how to use the TO_CHAR function to convert a date to a character string when we want to retrieve and display a date value in a non-default format.
• Here is a reminder of TO_CHAR:

```
SELECT first_name, TO_CHAR(hire_date,'Month, fmdd, yyyy')
FROM employees
WHERE employee_id = 101;
```

FIRST_NAME	TO_CHAR(HIRE_DATE,'MONTH,FMDD,YYYY')
Neena	September, 21, 1989

• Similarly, if we want to INSERT a row with a non-default format for a date column, we must use the TO_DATE function to convert the date value (a character string) to a date.

```
INSERT INTO copy_employees
        (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary)
VALUES (301,'Katie','Hernandez', 'khernandez','8586667641',
TO_DATE('July 8, 2015', 'Month fmdd, yyyy'), 'MK_REP',4200);
```

• A second example of TO_DATE allows the insertion of a specific time of day, overriding the default time of midnight.

```
INSERT INTO copy_employees
        (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary)
VALUES (303,'Angelina','Wright', 'awright','4159982010',
TO_DATE('July 10, 2015 17:20', 'Month fmdd, yyyy HH24:MI'), 'MK_REP', 3600);
```

```
SELECT first_name, last_name,
TO_CHAR(hire_date, 'dd-Mon-YYYY HH24:MI') As "Date and Time"
FROM copy_employees
WHERE employee_id = 303;
```

FIRST_NAME	LAST_NAME	Date and Time
Angelina	Wright	10-Jul-2015 17:20

Using A Subquery To Copy Rows

• Each INSERT statement we have seen so far adds only one row to the table.
• But suppose we want to copy 100 rows from one table to another.
• We do not want to have to write and execute 100 separate INSERT statements, one after the other.
• That would be very time-consuming.
• Fortunately, SQL allows us to use a subquery within an INSERT statement.
• All the results from the subquery are inserted into the table.
• So we can copy 100 rows – or 1000 rows – with one multiple-row subquery within the INSERT.
• As you would expect, you don't need a VALUES clause when using a subquery to copy rows because the inserted values will be exactly the values returned by the subquery.
• In the example shown, a new table called SALES_REPS is being populated with copies of some of the rows and columns from the EMPLOYEES table.
• The WHERE clause is selecting those employees that have job IDs like '%REP%'.

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

• The number of columns and their data types in the column list of the INSERT clause must match the number of columns and their data types in the subquery.
• The subquery is not enclosed in parentheses as is done with the subqueries in the WHERE clause of a SELECT statement.
• If we want to copy all the data – all rows and all columns – the syntax is even simpler.
• To select all rows from the EMPLOYEES table and insert them into the SALES_REPS table, the statement would be written as shown:

```
INSERT INTO sales_reps
SELECT *
FROM employees;
```

• Again, this will work only if both tables have the same number of columns with matching data types, and they are in the same order.

12-2 Updating Column Values and Deleting Rows

In this lesson, you will learn to:

- Construct and execute an UPDATE statement
- Construct and execute a DELETE statement
- Construct and execute a query that uses a subquery to update and delete data from a table
- Construct and execute a query that uses a correlated subquery to update and delete from a table
- Explain how foreign-key and primary-key integrity constraints affect UPDATE and DELETE statements
- Explain the purpose of the FOR UPDATE Clause in a SELECT statement
- Updating, inserting, deleting, and managing data is a Database Administrator's (DBA's) job.(nd !)

UPDATE

- The UPDATE statement is used to modify existing rows in a table.
- UPDATE requires four values:

– the name of the table
– the name of the column(s) whose values will be modified
– a new value for each of the column(s) being modified
– a condition that identifies which rows in the table will be modified.

- The new value for a column can be the result of a single-row subquery.
- The example shown uses an UPDATE statement to change the phone number of one employee in the employees table.
- Note that the copy_employees table is used in this transaction.

```
UPDATE copy_employees SET phone_number = '123456' WHERE employee_id = 303;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	PHONE_NUMBER
303	Angelina	Wright	123456

- We can change several columns and/or several rows in one UPDATE statement.
- This example changes both the phone number and the last name for two employees.

```
UPDATE copy_employees
SET phone_number = '654321', last_name = 'Jones'
WHERE employee_id >= 303;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	PHONE_NUMBER
303	Angelina	Jones	654321
304	Test	Jones	654321

- Take care when updating column values.
- If the WHERE clause is omitted, every row in the table will be updated.
- This may not be what was intended.

Updating a Column with a value from a Subquery

- We can use the result of a single-row subquery to provide the new value for an updated column.

```
UPDATE copy_employees
SET salary =
        (SELECT salary
        FROM copy_employees
        WHERE employee_id = 100)
WHERE employee_id = 101;
```

- This example changes the salary of one employee (id = 101) to the same salary as another employee (id = 100).

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000

101	Neena	Kochhar	24000

•As usual, the subquery executes first and retrieves the salary for employee id = 100.
•This salary value is then used to update the salary for employee id = 101.

Updating Two Columns with Two Subquery Statements
• To update several columns in one UPDATE statement, it is possible to write multiple single-row subqueries, one for each column.
• In the following example the UPDATE statement changes the salary and job id of one employee (id = 206) to the same values as another employee (id = 205).

```
UPDATE copy_employees
SET salary =
        (SELECT salary
        FROM copy_employees
        WHERE employee_id = 205),
    job_id =
        (SELECT job_id
        FROM copy_employees
        WHERE employee_id = 205)
WHERE employee_id = 206;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	JOB_ID
205	Shelley	Higgins	12000	AC_MGR
206	William	Gietz	12000	AC_MGR

Updating Rows Based On Another Table
• As you may have expected, the subquery can retrieve information from one table which is then used to update another table.
• In this example, a copy of the employees table was created.
• Then data from the original employees table was retrieved, copied, and used to populate the copy_employees table.

```
UPDATE copy_employees
SET salary =
        (SELECT salary
        FROM employees
        WHERE employee_id = 205)
WHERE employee_id = 202;
```

Updating Rows Using Correlated Subquery
• As you already know subqueries can be either stand alone or correlated.
• In a correlated subquery, you are updating a row in a table based on a select from that same table.
• In the example below, a copy of the department name column was created in the employees table.
• Then data from the original departments table was retrieved, copied, and used to populate the copy of the column in the employees table

```
ALTER TABLE copy_employees
ADD (department_name varchar2(30) NOT NULL);

UPDATE copy_employees e
SET e.department_name =
        (SELECT d.department_name
        FROM departments d
        WHERE e.department_id = d.department_id);
```

DELETE
• The DELETE statement is used to remove existing rows in a table.
• The statement requires two values:

– the name of the table
– the condition that identifies the rows to be deleted

• The example shown uses the copy employee table to delete one row—the employee with ID number 303.

```
DELETE from copy_employees
WHERE employee_id = 303;
```

• What do you predict will be deleted if the WHERE clause is eliminated in a DELETE statement?
• All rows in the table are deleted if you omit the WHERE clause.

Subquery DELETE

• Subqueries can also be used in DELETE statements.
• The example shown deletes rows from the employees table for all employees who work in the Shipping department.
• Maybe this department has been renamed or closed down.

```
DELETE FROM copy_employees
WHERE department_id =
        (SELECT department_id
        FROM departments
        WHERE department_name = 'Shipping');
```

5 row(s) deleted

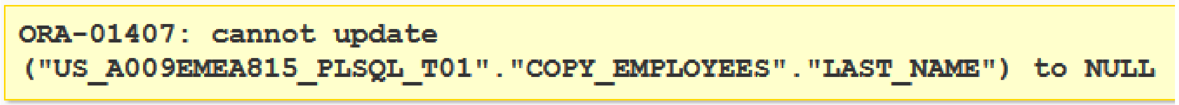
• The example below deletes rows of all employees who work for a manager that manages less than 2 employees.

```
DELETE FROM copy_employees e
 WHERE e.manager_id IN
        (SELECT d.manager_id
        FROM employees d
        HAVING count (d.department_id) < 2
        GROUP BY d.manager_id);
```

Integrity Constraint Errors

• Integrity constraints ensure that the data conforms to a needed set of rules.
• The constraints are automatically checked whenever a DML statement which could break the rules is executed.
• If any rule would be broken, the table is not updated and an error is returned.
• This example violates a NOT NULL constraint because last_name has a not null constraint and id=999 does not exist, so the subquery returns a null result.

```
UPDATE copy_employees
SET last_name =
        (SELECT last_name
        FROM copy_employees
        WHERE employee_id = 999)
WHERE employee_id = 101;
```



```
ORA-01407: cannot update
("US_A009EMEA815_PLSQL_T01"."COPY_EMPLOYEES"."LAST_NAME") to NULL
```

• When will primary key - foreign key constraints be checked?
• The EMPLOYEES table has a foreign key constraint on department_id which references the department_id of the DEPARTMENTS table.
• This ensures that every employee belongs to a valid department.
• In the DEPARTMENTS table, department_ids 10 and 20 exist but 15 does not.
• Which of the following statements will return an error?

1. UPDATE employees SET department_id = 15
   WHERE employee_id = 100;
2. DELETE FROM departments WHERE department_id = 10;

```
3. UPDATE employees SET department_id = 10
   WHERE department_id = 20;
```

• When modifying your table copies (for example copy_employees), you might see not null constraint errors, but you will not see any primary key – foreign key constraint errors.
• The reason for this is that the CREATE TABLE …. AS (SELECT …) statement that is used to create the copy of the table, copies both the rows and the not null constraints, but does not copy the primary key – foreign key constraints.
• Therefore, no primary key – foreign key constraints exist on any of the copied tables.
• Adding constraints is covered in another lesson.

FOR UPDATE Clause in a SELECT Statement
• When a SELECT statement is issued against a database table, no locks are issued in the database on the rows returned by the query you are issuing.
• Most of the time this is how you want the database to behave—to keep the number of locks issued to a minimum.
• Sometimes, however, you want to make sure no one else can update or delete the records your query is returning while you are working on those records.
• This is when the FOR UPDATE clause is used.
• As soon as your query is executed, the database will automatically issue exclusive row-level locks on all rows returned by your SELECT statement, which will be held until you issue a COMMIT or ROLLBACK command.
• Reminder: The on-line/hosted instance of APEX will autocommit, and the row-level lock will not occur.
• If you use a FOR UPDATE clause in a SELECT statement with multiple tables in it, all the rows from all tables will be locked.

```
SELECT e.employee_id, e.salary, d.department_name
FROM employees e JOIN departments d USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

• These four rows are now locked by the user who ran the SELECT statement until the user issues a COMMIT or ROLLBACK.

EMPLOYEE_ID	SALARY	DEPARTMENT_NAME
141	3500	Shipping
142	3100	Shipping
143	2600	Shipping
144	2500	Shipping

12-3 DEFAULT Values, MERGE, and Multi-Table Insert

In this lesson, you will learn to:
• Understand when to specify a DEFAULT value
• Construct and execute a MERGE statement
• Construct and execute DML statements using subqueries
• Construct and execute multi-table inserts
Purpose
• How do you determine what has been newly inserted or what has been recently changed?
• In this lesson, you will learn a more efficient method to update and insert data using a sequence of conditional INSERT and UPDATE commands in a single atomic statement.
• You will also learn how to retrieve data from one single subquery and INSERT the rows returned into more than one target table.
• As you extend your knowledge in SQL, you'll appreciate effective ways to accomplish your work.

DEFAULT
• Each column in a table can have a default value specified for it.
• In the event that a new row is inserted and no value for the column is assigned, the default value will be assigned instead of a null value.
• Using default values allows you to control where and when the default value should be applied.

• The default value can be a literal value, an expression, or a SQL function such as SYSDATE and USER, but the value cannot be the name of another column.
• The default value must match the data type of the column.
• DEFAULT can be specified for a column when the table is created or altered.
• The example below shows a default value being specified for the hire_date column at the time the table is created:

```
CREATE TABLE my_employees (
        hire_date DATE DEFAULT SYSDATE,
        first_name VARCHAR2(15),
        last_name VARCHAR2(15));
```

• When rows are added to this table, SYSDATE will be assigned to any row that does not explicitly specify a hire_date value.

Explicit DEFAULT with INSERT
• Explicit defaults can be used in INSERT and UPDATE statements.
• The INSERT example using the my_employees table shows the explicit use of DEFAULT:

```
INSERT INTO my_employees (hire_date, first_name, last_name)
VALUES (DEFAULT, 'Angelina','Wright');
```

• Implicit use of DEFAULT

```
INSERT INTO my_employees (first_name, last_name)
VALUES ('Angelina','Wright');
```

• Explicit defaults can be used in INSERT and UPDATE statements.
• The UPDATE example using the my_employees table shows explicit use of DEFAULT.

```
UPDATE my_employees
SET hire_date = DEFAULT
WHERE last_name = 'Wright';
```

• If a default value was specified for the hire_date column, the column is assigned the default value.
• However, if no default value was specified when the column was created, a null value is assigned.

MERGE
• Using the MERGE statement accomplishes two tasks at the same time. MERGE will INSERT and UPDATE simultaneously. If a value is missing, a new one is inserted.
• If a value exists but needs to be changed, MERGE will update it.
• To perform these kinds of changes to database tables, you need to have INSERT and UPDATE privileges on the target table and SELECT privileges on the source table.
• Aliases can be used with the MERGE statement.

MERGE Syntax
• One row at a time is read from the source table and compared to rows in the destination table using the matching condition.
• If a matching row exists in the destination table, the source row is used to update one or more columns in the matching destination row.
• If a matching row does not exist, values from the source row are used to insert a new row into the destination table.

```
MERGE INTO destination-table USING source-table ON matching-condition
WHEN MATCHED THEN UPDATE
        SET ……
WHEN NOT MATCHED THEN INSERT
VALUES (……);
```

• This example uses the EMPLOYEES table (alias e) as a data source to insert and update rows in a copy of the table named COPY_EMP (alias c).

```
MERGE INTO copy_emp c USING employees e ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN UPDATE
        SET
                c.last_name = e.last_name,
                c.department_id = e.department_id
WHEN NOT MATCHED THEN INSERT
VALUES (e.employee_id, e.last_name, e.department_id);
```

• EMPLOYEES rows 100 and 103 have matching rows in COPY_EMP, and so the matching COPY_EMP rows were updated.
• EMPLOYEE 142 had no matching row, and so was inserted into COPY_EMP.

EMPLOYEES (source table)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

COPY_EMP before the MERGE is executed

EMPLOYEE_ID	**LAST_NAME**	**DEPARTMENT_ID**
100	Smith	40
103	Chang	30

COPY_EMP after the MERGE has executed

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

Multi-Table Inserts
• Multi-table inserts can be unconditional or conditional. In an unconditional multi-table insert, Oracle will insert all rows returned by the subquery into all table insert clauses found in the statement.
• In a conditional multi-table insert, you can specify either ALL or FIRST.
• Specifying ALL:
– If you specify ALL, the default value, the database evaluates each WHEN clause regardless of the results of the evaluation of any other WHEN clause.
– For each WHEN clause whose condition evaluates to true, the database executes the corresponding INTO clause list.
• Specifying FIRST:
– If you specify FIRST, the database evaluates each WHEN clause in the order in which it appears in the statement.
– For the first WHEN clause that evaluates to true, the database executes the corresponding INTO clause and skips subsequent WHEN clauses for the given row.
• Specifying the ELSE clause:
• For a given row, if no WHEN clause evaluates to true, the database executes the INTO clause list associated with the ELSE clause.
• If you did not specify an else clause, then the database takes no action for that row.

Multi-Table Inserts Syntax
• Multi-table insert statement syntax is as follows:

```
INSERT ALL INTO clause VALUES clause SUBQUERY
```

•Multi-table insert statement example is as follows:

```
INSERT ALL
        INTO my_employees
                VALUES (hire_date, first_name, last_name)
        INTO copy_my_employees
                VALUES (hire_date, first_name, last_name)
```

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

Multi-Table Inserts Conditional

```
INSERT ALL
        WHEN call_ format IN ('tlk','txt','pic') THEN
        INTO all_calls
                VALUES (caller_id, call_timestamp, call_duration, call_format)
        WHEN call_ format IN ('tlk','txt') THEN
        INTO police_record_calls
                VALUES (caller_id, call_timestamp, recipient_caller)
        WHEN call_duration < 50 AND call_type = 'tlk' THEN
        INTO short_calls
                VALUES (caller_id, call_timestamp, call_duration)
        WHEN call_duration > = 50 AND call_type = 'tlk' THEN
        INTO long_calls
                VALUES (caller_id, call_timestamp, call_duration)
SELECT caller_id, call_timestamp, call_duration, call_format, recipient_caller
FROM calls
WHERE TRUNC(call_timestamp ) = TRUNC(SYSDATE);
```

13-1 Creating Tables

In this lesson, you will learn to:

- List and categorize the main database objects
- Review a table structure
- Describe how schema objects are used by the Oracle database
- Create a table using the appropriate data type for each column
- Explain the use of external tables
- Query the Data Dictionary to obtain the names and other attributes of database objects
- As a Database Administrator (DBA), you will be expected to know how to create tables as well.

Database Schema Objects

- An Oracle Database can contain many different types of objects.
- The main database object types are:
  - Table
  - Index
  - Constraint
  - View
  - Sequence
  - Synonym
- Some of these object types can exist independently and others can not.
- Database objects taking up significant storage space are known as Segments.
- Tables and Indexes are examples of Segments, as the values stored in the columns of each row take up significant physical disk space.
- Views, Constraints, Sequences, and Synonyms are also objects, but the only space they require in the database is in the definition of the object—none of them have any data rows associated with them.
- The database stores the definitions of all database objects in the Data Dictionary, and these definitions are accessible to all users of the database as well as to the database itself.
- Have you ever wondered how Oracle knows which columns to return from a Query?
- The database looks up the definition of the table used in the query, translates the '*' into the full list of columns, and returns the result to you.
- The database uses the Data Dictionary for all statements you issue, even if you list the column names instead of using '*'.
- It checks that the tables you are referencing in your statement exist in the database, it checks that the column names are correct, it checks if you have the correct privileges to perform the action you are requesting, and finally it uses the Data Dictionary to decide the Execution Plan – how it will actually perform the request.
- The Data Dictionary itself can be queried by all database users.
- In Application Express, it can be accessed both via SQL statements in the SQL Workshop> SQL Commands interface and also from the SQL Workshop> Object Browser interface.

Table Creation

- All data in a relational database is stored in tables.
- When creating a new table, use the following rules for table names and column names:
  - Must begin with a letter
  - Must be 1 to 30 characters long
  - Must contain only A - Z, a - z, 0 - 9, _ (underscore), $, and #
  - Must not duplicate the name of another object owned by the same user
  - Must not be an Oracle Server reserved word

Naming Conventions

- Table names are not case sensitive.
- For example, STUDENTS is treated the same as STuDents or students.
- Table names should be plural, for example STUDENTS, not student.
- Creating tables is part of SQL's data definition language (DDL).
- Other DDL statements used to set up, change, and remove data structures from tables include ALTER, DROP, RENAME, and TRUNCATE.

CREATE TABLE

- To create a new table, you must have the CREATE TABLE privilege and a storage area for it.

• The database administrator uses data control language (DCL) statements to grant this privilege to users and assign a storage area.
• Tables belonging to other users are not in your schema.
• If you want to use a table that is not in your schema, use the table owner's name as a prefix to the table name:
SELECT * FROM mary.students;
–You must be granted access to a table to be able to select from it.

CREATE TABLE Syntax
• To create a new table, use the following syntax details:
– table is the name of the table
– column is the name of the column
– Data type is the column's data type and length
– DEFAULT expression specifies a default value if a value is omitted in the INSERT statement

```
CREATE TABLE table
(column data type [DEFAULT expression],
(column data type [DEFAULT expression],
(……[ ] );
```

CREATE TABLE Example
• The examples below show the CREATE TABLE statement:

```
CREATE TABLE my_cd_collection
        (cd_number NUMBER(3),
        title VARCHAR2(20),
        artist VARCHAR2(20),
        purchase_date DATE DEFAULT SYSDATE);

CREATE TABLE my_friends
        (first_name VARCHAR2(20),
        last_name VARCHAR2(30),
        email VARCHAR2(30),
        phone_num VARCHAR2(12),
        birth_date DATE);
```

External Tables
• Oracle also supports another table type: External table.
• In an external table, the data rows are not held inside the database files but are instead found in a flat file, stored externally to the database.
• One of the many benefits for Oracle is that data held in external tables only has to be backed up once, and then never again unless the contents of the file change.
• The syntax to create an external table is very similar to that of creating a standard table, except that it has extra syntax at the end.
• The new syntax (shown in red) on the following slides, is not used in standard SQL statements for table creation.
• ORGANIZATION EXTERNAL -- tells Oracle to create an external table
• TYPE ORACLE_LOADER -- of type Oracle Loader (an Oracle Product)
• DEFAULT DIRECTORY def_dir1 -- the name of the directory for the file
• ACCESS PARAMETERS -- how to read the file
• RECORDS DELIMITED BY NEWLINE -- how to identify the start of a new row
• FIELDS – the field name and data type specifications
• LOCATION – name of the actual file containing the data
• n example of the new syntax is found in red on the next slide.

External Tables Example

```
CREATE TABLE emp_load
        (employee_number CHAR(5),
        employee_dob CHAR(20),
        employee_last_name CHAR(20),
        employee_first_name CHAR(15),
        employee_middle_name CHAR(15),
        employee_hire_date DATE)
ORGANIZATION EXTERNAL
        (TYPE ORACLE_LOADER
                DEFAULT DIRECTORY def_dir1
                ACCESS PARAMETERS
        (RECORDS DELIMITED BY NEWLINE
        FIELDS (employee_number CHAR(2),
                employee_dob CHAR(20),
                employee_last_name CHAR(18),
                employee_first_name CHAR(11),
                employee_middle_name CHAR(11),
                employee_hire_date CHAR(10) date_format DATE mask "mm/dd/yyyy"))
        LOCATION ('info.dat'));
```

Data Dictionary

• Two kinds of tables exists in an Oracle Database: User tables and Data Dictionary tables.
• You can issue SQL statements to access both kinds of tables—you can select, insert, update, and delete data in the user tables, and you can select data in the Data Dictionary tables.
• **Data Dictionary tables:**
– DICTIONARY, USER_OBJECTS, USER_TABLES, USER_SEGMENTS, USER_INDEXES, etc.
• The Data Dictionary tables are all owned by a special Oracle user called SYS and only SELECT statements should be used when working with any of these tables.
• If any Oracle user attempts to do inserts, updates, or deletes against any of the Data Dictionary tables, the operation is disallowed as it might compromise the integrity of the entire database.
• When you are using the Data Dictionary views in the SQL Commands interface, you need to know the names of the Dictionary views you are working with.
• In Oracle, this is quite simple: prefix the object type you are looking for with a USER_xxx or an ALL_xxx, where xxx is the object type.

```
SELECT table_name, status FROM USER_TABLES;
SELECT table_name, status FROM ALL_TABLES;
SELECT * FROM user_indexes;
SELECT * FROM user_objects WHERE object_type = 'SEQUENCE';
```

13-2 Using Data Types

In this lesson, you will learn to:
• Create a table using TIMESTAMP and TIMESTAMP WITH TIME ZONE column data types
• Create a table using INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND column data types
• Give examples of organizations and personal situations where it is important to know to which time zone a date-time value refers
• List and provide an example of each of the number, date, and character data types

Data Type Overview

• Each value manipulated by Oracle has a data type.
• A value's data type associates a fixed set of properties with the value.
• These properties cause the database to treat values of one data type differently from values of another data type.
• Different data types offer several advantages:
  – Columns of a single type produce consistent results.
  – For example, DATE data type columns always produce date values.
  – You cannot insert the wrong type of data into a column. For example, columns of data type DATE will prevent NUMBER type data from being inserted.
• For these reasons, each column in a relational database can hold only one type of data.
• You cannot mix data types within a column.

Common Data Types

• The most commonly used column data types for character and number values are below.

• For character values:
– CHAR (fixed size, maximum 2000 characters)
– VARCHAR2 (variable size, maximum 4000 characters)
– CLOB (variable size, maximum 128 terabytes)
• For number values:
– NUMBER (variable size, maximum precision 38 digits)
• The most commonly used column data types for date, time, and binary values are below.
• For date and time values:
  – DATE
  – TIMESTAMP ….
  – INTERVAL
• For binary values (eg. multimedia: JPG, WAV, MP3, and so on):
  – RAW (variable size, maximum 2000 bytes)
  – BLOB (variable size, maximum 128 terabytes)
• For character values, it is usually better to use VARCHAR2 or CLOB than CHAR, because it saves space.
• Number values can be negative as well as positive. For example, NUMBER(6,2) can store any value from +9999.99 down to –9999.99.

DATE-TIME Data Types

• The DATE data type stores a value of centuries down to whole seconds, but cannot store fractions of a second.
• '21-Aug-2003 17:25:30' is a valid value, but '21-Aug-2003 17:25:30.255' is not.
• The TIMESTAMP data type is an extension of the DATE data type which allows fractions of a second.
• For example, TIMESTAMP(3) allows 3 digits after the whole seconds, allowing values down to milliseconds to be stored.
• **TIMESTAMP example:**

```
CREATE TABLE time_ex1 (exact_time TIMESTAMP);

INSERT INTO time_ex1 VALUES ('10-Jun-2015 10:52:29.123456');

INSERT INTO time_ex1 VALUES (SYSDATE);
INSERT INTO time_ex1 VALUES (SYSTIMESTAMP);

SELECT * FROM time_ex1;

EXACT_TIME
10-JUN-15 10.52.29.123456 AM
16-JUL-15 08.17.08.000000 AM
16-JUL-15 08.17.16.610293 AM
```

TIMESTAMP…With [LOCAL] Time Zone

• Think about the time value '17:30'. Of course it means "half past five in the afternoon".
• But in which time zone?
• TIMESTAMP WITH TIME ZONE stores a time zone value as a displacement from Universal Coordinated Time or UCT (previously known as Greenwich Mean Time or GMT).
• A value of '21-Aug-2003 08:00:00 –5:00' means 8:00 am 5 hours behind UTC.
• This is US Eastern Standard Time (EST).

TIMESTAMP WITH TIME ZONE example:

```
CREATE TABLE time_ex2 (time_with_offset TIMESTAMP WITH TIME ZONE);

INSERT INTO time_ex2 VALUES (SYSTIMESTAMP);

INSERT INTO time_ex2 VALUES ('10-Jun-2015 10:52:29.123456 AM +2:00');

SELECT * FROM time_ex2;

TIME_WITH_OFFSET
16-JUL-15 08.49.47.126056 AM -07:00
```

10-JUN-15 10.52.29.123456 AM +02:00

• TIMESTAMP WITH LOCAL TIME ZONE is similar, but with one difference: when this column is selected in a SQL statement, the time is automatically converted to the selecting user's time zone.

• TIMESTAMP With…Time Zone Example:

```
CREATE TABLE time_ex3
        ( first_column TIMESTAMP WITH TIME ZONE,
        second_column TIMESTAMP WITH LOCAL TIME ZONE);

INSERT INTO time_ex3 (first_column, second_column)
VALUES ('15-Jul-2015 08:00:00 AM -07:00', '15-Nov-2007 08:00:00');
```

• Both values are stored with a time displacement of –07:00 hours (PST).
• But now a user in Istanbul executes:

```
SELECT * FROM time_ex3;
```

FIRST_COLUMN	SECOND_COLUMN
15-JUL-15 08.00.00.000000 AM -07:00	15-NOV-07 05.00.00.000000 PM

• Istanbul time is 9 hours ahead of PST; when it's 8am in Los Angeles, it's 5pm in Istanbul.

INTERVAL Data Types
• These store the elapsed time, or interval of time, between two date-time values.
• INTERVAL YEAR TO MONTH stores a period of time measured in years and months.
• INTERVAL DAY TO SECOND stores a period of time measured in days, hours, minutes, and seconds.

INTERVAL YEAR…TO MONTH
• Syntax:
INTERVAL YEAR [(year_precision)] TO MONTH
• year_precision is the maximum number of digits in the YEAR element.
•The default value of year_precision is 2.

INTERVAL YEAR…TO MONTH
• This example shows INTERVAL YEAR TO MONTH:

```
CREATE TABLE time_ex4
        (loan_duration1 INTERVAL YEAR(3) TO MONTH,
        loan_duration2 INTERVAL YEAR(2) TO MONTH);

INSERT INTO time_ex4 (loan_duration1, loan_duration2)
VALUES (INTERVAL '120' MONTH(3),
INTERVAL '3-6' YEAR TO MONTH);
```

Assume today's date is: 17-Jul-2015

```
SELECT SYSDATE + loan_duration1 AS "120 months from now",
SYSDATE + loan_duration2 AS "3 years 6 months from now"
FROM time_ex4;
```

120 months from now	3 years 6 months from now
17-Jul-2025	17-Jan-2019

INTERVAL DAY…TO SECOND
• This example shows interval DAY TO SECOND:

```
CREATE TABLE time_ex5
```

```
        (day_duration1 INTERVAL DAY(3) TO SECOND,
        day_duration2 INTERVAL DAY(3) TO SECOND);

INSERT INTO time_ex5 (day_duration1, day_duration2)
VALUES (INTERVAL '25' DAY(2), INTERVAL '4 10:30:10' DAY TO SECOND);

SELECT SYSDATE + day_duration1 AS "25 Days from now",
        TO_CHAR(SYSDATE + day_duration2, 'dd-Mon-yyyy hh:mi:ss')
        AS "precise days and time from now"
FROM time_ex5;
```

13-3 Modifying a Table

In this lesson, you will learn to:
• Explain why it is important to be able to modify a table
• Explain and provide an example for each of the DDL statements—ALTER, DROP, RENAME, and TRUNCATE—and the effect each has on tables and columns
• Construct a query and execute the ALTER TABLE commands ADD, MODIFY, and DROP
• Explain and perform FLASHBACK QUERY on a table
• Explain and perform FLASHBACK table operations
• Track the changes to data over a period of time
• Explain the rationale for using TRUNCATE versus DELETE for tables
• Add a comment to a table using the COMMENT ON TABLE command
• Name the changes that can and cannot be made to modify a column
• Explain when and why the SET UNUSED statement is advantageous

ALTER TABLE

• ALTER TABLE statements are used to:
  – Add a new column
  – Modify an existing column
  – Define a DEFAULT value for a column
  – Drop a column
• You can add or modify a column in a table, but you cannot specify where the column appears.
• Also, if a table already has rows of data and you add a new column to the table, the new column is initially null for all of the pre-existing the rows.

ALTER TABLE: Adding a Column

• To add a new column, use the SQL syntax shown:

```
ALTER TABLE tablename
        ADD (column name data type [DEFAULT expression],
        column name data type [DEFAULT expression], ...
```

•For example:

```
ALTER TABLE my_cd_collection
        ADD (release_date DATE DEFAULT SYSDATE);
ALTER TABLE my_friends
        ADD (favorite_game VARCHAR2(30));
```

ALTER TABLE: Modifying a Column

• Modifying a column can include changes to a column's data type, size, and DEFAULT value.
• Rules and restrictions when modifying a column are:
  – You can increase the width or precision of a numeric column.
  – You can increase the width of a character column.
  – You can decrease the width of a NUMBER column if the column contains only null values, or if the table has no rows.
  – For VARCHAR types, you can decrease the width down to the largest value contained in the column.
• You can change the data type only if the column contains null values.
• You can convert a CHAR column to VARCHAR2 or convert a VARCHAR2 COLUMN to CHAR only if the column contains null values, or if you do not change the size to something smaller than any value in the column.

• A change to the DEFAULT value of a column affects only later insertions to the table.

ALTER TABLE: Modifying a Column Example
nd: there are some mistekes in slide 13.
• Example: a table has been created with two columns:

```
CREATE TABLE mod_emp
        (last_name VARCHAR2(20),
        salary NUMBER(8,2));
```

• Which of these modification would be allowed, and which would not? (Consider your answers both with and without rows of data in the table.)

ALTER TABLE mod_emp MODIFY (last_name VARCHAR2(30));
•Would be permitted with or without data as column width increased.

ALTER TABLE mod_emp MODIFY (last_name VARCHAR2(10));
• Would be permitted only if columns were empty, or the largest name was 10 or less characters

ALTER TABLE mod_emp MODIFY (salary NUMBER(10,2));
• Would be permitted with or without data as column precision increased.

ALTER TABLE mod_emp MODIFY (salary NUMBER(8,2) DEFAULT 50);
•Would be permitted with or without data as only a DEFAULT value added.

ALTER TABLE: Dropping a Column
• When dropping a column the following rules apply:
  – A column containing data may be dropped.
  – Only one column can be dropped at a time.
  – You can't drop all of the columns in a table; at least one column must remain.
  – Once a column is dropped, the data values in it cannot be recovered.
• SQL Syntax:
ALTER TABLE tablename DROP COLUMN column name;

•For Example:
ALTER TABLE my_cd_collection DROP COLUMN release_date;

ALTER TABLE my_friends DROP COLUMN favorite_game;

SET UNUSED Columns
• Dropping a column from a large table can take a long time.
• A quicker alternative is to mark the column as unusable.
• The column values remain in the database but cannot be accessed in any way, so the effect is the same as dropping the column.
• In fact, you could add a new column to the database with the same name as the unused column.
• The unused columns are there, but invisible!
• Syntax:
ALTER TABLE tablename SET UNUSED (column name);

SET UNUSED Columns Example
• Example:
ALTER TABLE copy_employees
SET UNUSED (email)

•DROP UNUSED COLUMNS removes all columns currently marked as unused.
•You use this statement when you want to reclaim the extra disk space from unused columns in a table.
•Example:
ALTER TABLE copy_employees DROP UNUSED COLUMNS;

ALTER TABLE Summarized

• This cart summarizes the uses of the ALTER TABLE command:

Syntax	**Outcomes**	**Concerns**
ALTER TABLE tablename ADD (column name data type [DEFAULT expression], column name data type [DEFAULT expression], ...	Adds a new column to a table	You cannot specify where the column is to appear in the table. It becomes the last column.
ALTER TABLE tablename MODIFY (column name data type [DEFAULT expression], column name data type, …	Used to change a column's data type, size, and default value	A change to the default value of a column affects only subsequent insertions to the table.
ALTER TABLE tablename DROP COLUMN column name;	Used to drop a column from a table	The table must have at least one column remaining in it after it is altered. Once dropped, the column cannot be recovered.
ALTER TABLE tablename SET UNUSED (column name);	Used to mark one or more columns so they can be dropped later	Does not restore disk space. Columns are treated as if they were dropped.
ALTER TABLE tablename DROP UNUSED COLUMNS	Removes from the table all columns currently marked as unused	Once set unused, there is no access to the columns; no data displayed using DESCRIBE. Permanent removal; no rollback.

DROP TABLE

• The DROP TABLE statement removes the definition of an Oracle table.
• The database loses all the data in the table and all the indexes associated with it.
• When a DROP TABLE statement is issued:
  – All data is deleted from the table.
  – The table's description is removed from the Data Dictionary.
• The Oracle Server does not question your decision and it drops the table immediately.
• In the next slide, you will see that you may be able to restore a table after it is dropped, but it is not guaranteed.
• Only the creator of the table or a user with DROP ANY TABLE privilege (usually only the DBA) can remove a table.
• Syntax: DROP TABLE tablename;
• Example: DROP TABLE copy_employees;

FLASHBACK TABLE

• If you drop a table by mistake, you may be able to bring that table and its data back.
• Each database user has his own recycle bin into which dropped objects are moved, and they can be recovered from here with the FLASHBACK TABLE command.
• This command can be used to restore a table, a view, or an index that was dropped in error.
• The Syntax is:
FLASHBACK TABLE tablename TO BEFORE DROP;

• For example, if you drop the EMPLOYEES table in error, you can restore it by simply issuing the command:
FLASHBACK TABLE copy_employees TO BEFORE DROP;
• As the owner of a table, you can issue the flashback command, and if the table that you are restoring had any indexes, then these are also restored.
• It is possible to see which objects can be restored by querying the data dictionary view USER_RECYCLEBIN.
• The USER_RECYCLEBIN view can be queried like all other data dictionary views:

```
SELECT original_name, operation, droptime
FROM user_recyclebin
```

ORIGINAL_NAME	**OPERATION**	**DROPTIME**
EMPLOYEES	DROP	2007-12-05:12.34.24
EMP_PK	DROP	2007-12-05:12.34.24

• Once a table has been restored by the FLASHBACK TABLE command, it is no longer visible in the USER_RECYCLEBIN view.
• Any indexes that were dropped with the original table will also be restored.

• It may be necessary (for security reasons) to completely drop a table, bypassing the recycle bin.
• This can be done by adding the keyword PURGE.
DROP TABLE copy_employees PURGE;

RENAME
• To change the name of a table, use the RENAME statement.
• This can be done only by the owner of the object or by the DBA.
• Syntax:
RENAME old_name to new_name;
•Example:
RENAME my_cd_collection TO my_music;
•We will see later that we can rename other types of objects such as views, sequences, and synonyms.
RENAME old_name to new_name;

TRUNCATE
• Truncating a table removes all rows from a table and releases the storage space used by that table.
• When using the TRUNCATE TABLE statement:
  – You cannot roll back row removal.
  – You must be the owner of the table or have been given DROP ANY TABLE system privileges.
• Syntax: TRUNCATE TABLE tablename;
• The DELETE statement also removes rows from a table, but it does not release storage space.
• TRUNCATE is faster than DELETE because it does not generate rollback information.

COMMENT ON TABLE
• You can add a comment of up to 2,000 characters about a column, table, or view by using the COMMENT statement.

• Syntax:
COMMENT ON TABLE tablename | COLUMN table.column
IS 'place your comment here';

• Example:
COMMENT ON TABLE employees
IS 'Western Region only';

• To view the table comments in the data dictionary:
SELECT table_name, comments
FROM user_tab_comments;

TABLE_NAME	COMMENTS
EMPLOYEES	Western Region Only

• If you want to drop a comment previously made on a table or column, use the empty string("):
COMMENT ON TABLE employees IS ' ' ;

FLASHBACK QUERY
• Luckily, Oracle has a facility that allows you to view row data at specific points in time, so you can compare different versions of a row over time.
• You can use the FLASHBACK QUERY facility to examine what the rows looked like BEFORE those changes were applied.
• When Oracle changes data, it always keeps a copy of what the amended data looked like before any changes were made.
• So it keeps a copy of the old column value for a column update, it keeps the entire row for a delete, and it keeps nothing for an insert statement.
• These old copies are held in a special place called the UNDO tablespace.
• Users can access this special area of the Database using a flashback query.
• You can look at older versions of data by using the VERSIONS clause in a SELECT statement.
• For example:

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
        versions_operation AS "OPERATION",
        versions_starttime AS "START_DATE",
        versions_endtime AS "END_DATE", salary
FROM employees
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 1;
```

•The SCN number referred to in the above query means the System Change Number and is a precise identification of time in the database.
• It is a sequential number incremented and maintained by the database itself.
• The contents are as follows for employee_id 1 in the employees table.

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
        versions_operation AS "OPERATION",
        versions_starttime AS "START_DATE", versions_endtime AS "END_DATE", salary
FROM copy_employees
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 1;
no data found
```

• Then we create the employee:

```
INSERT INTO copy_employees
VALUES (1, 'Natacha', 'Hansen', 'NHANSEN', '4412312341234',
'07-SEP-1998', 'AD_VP', 12000, null, 100, 90, NULL);
```

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
        versions_operation AS "OPERATION",
        versions_starttime AS "START_DATE", versions_endtime AS "END_DATE", salary
FROM copy_employees
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 1;
```

EMPLOYEE_ID	NAME	OPERATION	START_DATE	END_DATE	SALARY
1	Natacha Hansen	I	07-SEP-1998 06.51.58 AM	-	12000

• Then you can update the row:

```
UPDATE copy_employees SET salary = 1 WHERE employee_id = 1;
```

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
        versions_operation AS "OPERATION",
        versions_starttime AS "START_DATE", versions_endtime AS "END_DATE", salary
FROM copy_employees
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 1;
```

EMPLOYEE_ID	NAME	OPERATION	START_DATE	END_DATE	SALARY
1	Natacha Hansen	U	07-SEP-1998 06.57.01 AM	-	1
1	Natacha Hansen	I	07-SEP-1998 06.51.58 AM	07-SEP-1998 06.57.01 AM	12000

• Then you can delete the row:

```
DELETE from copy_employees
WHERE employee_id = 1;
```

```
SELECT employee_id,first_name ||' '|| last_name AS "NAME",
        versions_operation AS "OPERATION",
        versions_starttime AS "START_DATE",
        versions_endtime AS "END_DATE", salary
FROM copy_employees
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 1;
```

EMPLOYEE_ID	NAME	OPERATION	START_DATE	END_DATE	SALARY
1	Natacha Hansen	D	07-SEP-1998 07.00.10 AM	-	1
1	Natacha Hansen	U	07-SEP-1998 06.57.01 AM	07-SEP-1998 07.00.10 AM	1
1	Natacha Hansen	I	07-SEP-1998 06.51.58 AM	07-SEP-1998 06.57.01 AM	12000

The result from the last query on the previous slide is only available when using Flashback query, i.e. the VERSIONS clause.

• If you attempt a normal search from employee_id = 1 following the delete statement, you would have received the normal error, No Data Found.

```
SELECT employee_id, salary
FROM copy_employees
WHERE employee_id = 1;
```