Chapter 4 (Data Manipulation (inserts))

```
INSERT INTO tablename [(columnname, …)]
VALUES (datavalue, …);
```

"The reason Las Vegas is so crowded is that no one has the plane fare to leave."

The keywords INSERT INTO are followed by the name of the table into which rows will be entered. The table name is followed by a list of the columns containing the data. The VALUES clause identifies the data values to be inserted in the table. You list the data values in parentheses after the VALUES keyword.

If the data entered in the VALUES clause contains a value for every column and is in the same order as columns in the table. Column names can be omitted in the INSERT INTO clause.

If you enter data for only some columns, or if columns are listed in a different order than they're listed in the table, the column names must be provided in the INSERT INTO clause in the same order as they're given in the VALUES clause. You must list the column names inside parentheses after the table name in the INSERT INTO clause.

If more than one column is listed, column names must be separated by commas. If more than one data value is entered, the values must be separated by commas.

You must use single quotes to enclose non-numeric data inserted in a column.

You can take one of the following approaches to indicate that a column contains a NULL value (indicating an absence of data):

- List all columns except the one that will not have any value in the INSERT INTO clause, and provide data for the listed columns in the VALUES clause.

- In the VALUES clause, substitute two single quotes in the position that should contain the account manager's assigned region. Oracle interprets the two single quotes to mean that a NULL value should be stored in the column. Be sure you don't add a blank space between these single quotes, however. Doing so adds a blank space value rather than a NULL value.

- In the VALUES clause, include the keyword NULL in the position where the region should be listed. As long as the keyword NULL isn't enclosed in single quotes, Oracle leaves the column blank. However, if the keyword is mistakenly entered as ' NULL', Oracle tries to store the word " NULL" in the column.

4.1 Inserting text

Example 4.1a (Inserting textual data)

To determine the number of characters in a string, you can use the LENGTH function.

```
DROP TABLE char_test;

--VARCHAR does not pad with spaces whereas CHAR is fixed  and is padded
--with spaces if enough characters are not provided. Keep in mind that space is data in that
--"HI " is different from "  HI".
CREATE  TABLE char_test
(
    cola VARCHAR(3),
    colb CHAR,
    colc CHAR(2)
);

DESC char_test;

--Have to have a value for every column in the table. The order of the values must match
--the order in which the columns appeared in the create table statement. Can use the
-- DESC  TABLE _name statement to find out what the proper order is.
INSERT  INTO  char_test VALUES  ('aaa','b','cc');
SELECT * FROM char_test;

-- Can identify the columns, which means that the order can be different.
INSERT  INTO  char_test (cola,colb,colc) VALUES
('aaa','b','cc');
SELECT * FROM char_test;

-- Can identify the columns in different order
INSERT  INTO  char_test (colb,colc,cola) VALUES
('b','cc','aaa');
```

```
-- Can identify fewer columns as long as a constraint is not violated
INSERT  INTO  char_test (colb,colc) VALUES  ('b','cc');

SELECT * FROM char_test;

table CHAR_TEST dropped.
table CHAR_TEST created.
DESC char_test
Name Null Type
---- ---- -----------
COLA      VARCHAR2(3)
COLB      CHAR(1)
COLC      CHAR(2)

1 rows inserted.
COLA COLB COLC
---- ---- ----
aaa  b    cc

1 rows inserted.
COLA COLB COLC
---- ---- ----
aaa  b    cc
aaa  b    cc

1 rows inserted.
1 rows inserted.
COLA COLB COLC
---- ---- ----
aaa  b    cc
aaa  b    cc
aaa  b    cc
     b    cc
```

```
-- Invalid:  cola  can be equal or less than three characters but not more.
INSERT  INTO  char_test (cola) VALUES  ('aaaa');

-- cola VARCHAR(3) can be less and will not pad it with spaces
INSERT  INTO  char_test (cola) VALUES  ('aa');
SELECT cola, LENGTH(cola), colb, colc FROM char_test;

--Can be NULL. Use the word NULL or single quotes with no spaces in between
INSERT  INTO  char_test VALUES  (NULL,'','c');
SELECT * FROM char_test;

-- colc char(2) blank padded
INSERT  INTO  char_test (colc) VALUES  ('c');

--To confirm the space, we can use the length function and feed into it colc which
--will tell us how many characters make up the data.
SELECT colc, LENGTH(colc) FROM char_test;
```

```
INSERT  INTO  char_test (cola) VALUES  ('aaaa')
Error report -
SQL Error: ORA-12899: value too large for column "IRAJ"."CHAR_TEST"."COLA" (actual: 4, maximum: 3)
12899. 00000 -  "value too large for column %s (actual: %s, maximum: %s)"
*Cause:    An attempt was made to insert or update a column with a value
           which is too wide for the width of the destination column.
           The name of the column is given, along with the actual width
           of the value, and the maximum allowed width of the column.
           Note that widths are reported in characters if character length
           semantics are in effect for the column, otherwise widths are
           reported in bytes.
*Action:   Examine the SQL statement for correctness.  Check source
           and destination column data types.
           Either make the destination column wider, or use a subset
           of the source column (i.e. use substring).
1 rows inserted.
COLA LENGTH(COLA) COLB COLC
---- ------------ ---- ----
aaa             3 b    cc
aaa             3 b    cc
aaa             3 b    cc
                  b    cc
aa              2

1 rows inserted.
COLA COLB COLC
---- ---- ----
aaa  b    cc
aaa  b    cc
aaa  b    cc
     b    cc
aa
          c
```

```
DELETE FROM char_test;

-- cola  can be equal or less than three characters but not more. Invalid
INSERT  INTO  char_test (cola) VALUES  ('aaaa');

-- cola VARCHAR(3) can be less and will not pad it with spaces
INSERT  INTO  char_test (cola) VALUES  ('aa');
SELECT cola, LENGTH(cola), colb, colc FROM char_test;

--can be NULL. Use the word NULL or single quotes with no spaces in between
INSERT  INTO  char_test VALUES  (NULL,'','c');
SELECT * FROM char_test;

-- colc char(2) blank padded
INSERT  INTO  char_test (colc) VALUES  ('c');

--To confirm the space, we can use the length function and feed into it.
--colc which will tell us how many characters make up the data.
SELECT colc, LENGTH(colc) FROM char_test;
```

```
Error starting at line : 3 in command -
INSERT  INTO  char_test (cola) VALUES  ('aaaa')
Error report -
SQL Error: ORA-12899: value too large for column "IRAJ"."CHAR_TEST"."COLA" (actual: 4, maximum:
12899. 00000 -  "value too large for column %s (actual: %s, maximum: %s)"
*Cause:    An attempt was made to insert or update a column with a value
           which is too wide for the width of the destination column.
           The name of the column is given, along with the actual width
           of the value, and the maximum allowed width of the column.
           Note that widths are reported in characters if character length
           semantics are in effect for the column, otherwise widths are
           reported in bytes.
*Action:   Examine the SQL statement for correctness.  Check source
           and destination column data types.
           Either make the destination column wider, or use a subset
           of the source column (i.e. use substring).
1 rows inserted.
COLA LENGTH(COLA) COLB COLC
---- ------------ ---- ----
aa              2

1 rows inserted.
COLA COLB COLC
---- ---- ----
aa
          c

1 rows inserted.
COLC LENGTH(COLC)
---- ------------

c               2
c               2
```

4.2 Inserting numbers

Example 4.2a (Inserting numerical data)

Single should not be used for numeric values in the INSERT INTO statement, such as the salary and commission values. Also, no formatting characters, such as a comma or dollar sign, should be included in values because they raise an error. A NUMBER column stores only digits. You add formatting characters when querying numeric values.

```
DROP TABLE number_test;
--The NUMBER data type can store real or whole numbers
--NUMBER(5) will store whole numbers that don't exceed five digits. It will round real numbers
--NUMBER(5,3): Can store a real number.  Two digits can be a whole number and three digits
--can be after the decimal point. If there are more digits after the decimal point then it
--rounds. If the whole number part is more than two digits, then it will give you an error
CREATE  TABLE  number_test
(
   col0 NUMBER,
   col1 NUMBER(5),
   col2 NUMBER(5,3)
);

-- With number you can store integers or floats.
INSERT  INTO  number_test VALUES  (12.2345, 2345, 23.253);
SELECT * FROM number_test;

-- Can put in NULLs using the NULL keyword or single quotes with no spaces
INSERT  INTO  number_test VALUES  (NULL,'',NULL);
SELECT * FROM number_test;

-- Can be less.
INSERT  INTO  number_test VALUES  (123,23,2);
SELECT * FROM number_test;

INSERT  INTO  number_test (col0) VALUES  (1234556);
INSERT  INTO  number_test (col0) VALUES  (12.34566);
SELECT * FROM number_test;

-- col1 number(5). More than five  digits is invalid.
INSERT  INTO  number_test (col1) VALUES  (123456);
--Will round to whole number
INSERT  INTO  number_test (col1) VALUES  (12.3456);
SELECT * FROM number_test;
-- Rounds up. Only looks at the first number after the decimal point  for rounding whole numbers.
INSERT  INTO  number_test (col1) VALUES  (12.5);
INSERT  INTO  number_test (col1) VALUES  (12.45);
SELECT * FROM number_test;
-- col2 number(5,3). Will round the significant digits to 3 digits.
INSERT  INTO  number_test (col2) VALUES  (12.2545);
SELECT * FROM number_test;

-- col2 number(5,3) is invalid. Can only have two digits to the left of  decimal point.
INSERT  INTO  number_test (col2) VALUES  (1234.2);

-- col2 number(5,3) invalid. Can only have two digits to the left of  decimal point.
INSERT  INTO  number_test (col2) VALUES  (123);

-- col2 number(5,3) will round the significant digits to 23.235
INSERT  INTO  number_test (col2) VALUES  (23.23456);
SELECT * FROM number_test;

-- INVALID: Goes beyond the two digits for the whole number part.
INSERT  INTO  number_test (col2) VALUES  (99.9999);
```

```
SQL> CREATE  TABLE  number_test
  2  (
  3  col0 NUMBER,
  4  col1 NUMBER(5),
  5  col2 NUMBER(5,3)
  6  );

Table created.

SQL>
SQL> -- With number you can store integers or floats.
SQL> INSERT  INTO  number_test VALUES  (12.2345, 2345, 23.253);

1 row created.

SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253

SQL>
SQL> -- Can put in nulls using the null keyword or single quotes with no spaces
SQL> INSERT  INTO  number_test VALUES  (null,'',null);

1 row created.

SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253


SQL>
SQL> -- Can be less.
SQL> INSERT  INTO  number_test VALUES  (123,23,2);

1 row created.
```

```
SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253

       123         23          2

SQL>
SQL> INSERT  INTO  number_test (col0) VALUES  (1234556);

1 row created.

SQL> INSERT  INTO  number_test (col0) VALUES  (12.34566);

1 row created.

SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253

       123         23          2
   1234556
  12.34566

SQL>
SQL> -- col1 number(5). More than five  is invalid
SQL> INSERT  INTO  number_test (col1) VALUES  (123456);
INSERT  INTO  number_test (col1) VALUES  (123456)
                                         *
ERROR at line 1:
ORA-01438: value larger than specified precision allowed for this column


SQL>
SQL> -- will round to whole number
SQL> INSERT  INTO  number_test (col1) VALUES  (12.3456);

1 row created.
```

```
SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253

       123         23          2
   1234556
  12.34566
                    12

6 rows selected.

SQL>
SQL> -- Rounds up. only looks at the first number after the decimal point
SQL> -- for rounding whole numbers
SQL> INSERT  INTO  number_test (col1) VALUES  (12.5);

1 row created.

SQL> INSERT  INTO  number_test (col1) VALUES  (12.45);

1 row created.

SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253

       123         23          2
   1234556
  12.34566
                    12
                    13
                    12

8 rows selected.

SQL>
SQL> -- col2 number(5,3). Will round the significant digits to 3 digits
SQL> INSERT  INTO  number_test (col2) VALUES  (12.2545);

1 row created.
```

```
SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253

       123         23          2
   1234556
  12.34566
                    12
                    13
                    12
                            12.255

9 rows selected.

SQL>
SQL> -- col2 number(5,3) invalid. can only have two digits to the left of
SQL> -- decimal point. This would be invalid
SQL> INSERT  INTO  number_test (col2) VALUES  (1234.2);
INSERT  INTO  number_test (col2) VALUES  (1234.2)
                                         *
ERROR at line 1:
ORA-01438: value larger than specified precision allowed for this column


SQL>
SQL> -- col2 number(5,3) invalid. can only have two digits to the left of
SQL> -- decimal point
SQL> INSERT  INTO  number_test (col2) VALUES  (123);
INSERT  INTO  number_test (col2) VALUES  (123)
                                         *
ERROR at line 1:
ORA-01438: value larger than specified precision allowed for this column


SQL>
SQL> -- col2 number(5,3) will round the significant digits to 23.235
SQL> INSERT  INTO  number_test (col2) VALUES  (23.23456);

1 row created.

SQL> SELECT * FROM number_test;

      COL0       COL1       COL2
---------- ---------- ----------
   12.2345       2345     23.253

       123         23          2
   1234556
  12.34566
                    12
                    13
                    12
                            12.255
                            23.235

10 rows selected.

SQL>
SQL> -- INVALID: goes beyond the two digits for the whole number
SQL> INSERT  INTO  number_test (col2) VALUES  (99.9999);
INSERT  INTO  number_test (col2) VALUES  (99.9999)
                                         *
ERROR at line 1:
ORA-01438: value larger than specified precision allowed for this column
```

4.3 Inserting Dates

The date value uses the default date format for Oracle: two- digit day, three- character-month, and two-digit year, separated by hyphens. (You can also provide a four- digit year value.)

The syntax of the TO_ DATE function is TO_ DATE(' d', ' f'), where d represents the date entered by the user, and f is the format-ting instruction for the date.

The TO_ CHAR function is often used to convert dates and numbers to a formatted character string. It's the opposite of the TO_ DATE function for handling date data discussed previously. The TO_ DATE function allows you to enter a date in any type of format and use the format argument to read the value as a date. The TO_ CHAR function, on the other hand, is used to have Oracle display dates in a particular format. The syntax of the TO_ CHAR function is TO_ CHAR(n, ' f'), where n is the date or number to format, and f is the format-ting instruction to use.

Date	**Description**	**Example**
Month	Name of the month spelled out	APRIL
MON	Three-letter abbreviation for the name of the month	APR
MM	Two-digit numeric value for the month	04
RM	Roman numeral representing the month	IV
D	Numeric value for the day of the week	4 (indicates Wednesday)
DD	Numeric value for the day of the month	28
DDD	Numeric value for day of the year	365 (indicates December 31)
DAY	Name of day of the week	WEDNESDAY
DY	Three-letter abbreviation for day of the week	WED
YYYY	Four digit numeric value for the year	2009
YYY or YY or Y	Numeric value for the last three, two, or single digit of the year	009, 09, or 9
YEAR	Spelled-out version of the year	TWO THOUSAND NINE
BC or AD	Value indicating B.C. or A.D.	2009 A.D.

Time	Description	Example
SS	Seconds	0-59
SSSS	Seconds past midnight	0-86399
MI	Minutes	0-59
HH or HH12	Hours	12
HH24	Hours (for military time)	0-23
A.M. or P.M.	Value indicating morning or evening hours	A.M. or P.M

Example 4.3a (Inserting dates)

```
CREATE  TABLE  date_test
(  col DATE  );

--Default date format "dd mon yy" or four digit year can be separated with spaces or dashes
--What is inserted is both date and time
INSERT  INTO  date_test VALUES ('01 feb 11');
INSERT  INTO  date_test VALUES  ('01-feb-11');

--This will be inserted as year 2045.
INSERT  INTO  date_test VALUES  ('01-feb-45');

INSERT  INTO  date_test VALUES  ('01-feb-1945');

--Insert the current date and time.
INSERT  INTO  date_test VALUES  (SYSDATE);

--The time is not visible but it is in there.
SELECT * FROM date_test;
```

```
SQL> CREATE  TABLE  date_test
  2  (
  3    col DATE
  4  );

Table created.

SQL>
SQL> --default date format mm dd yy or four digit year
SQL> --can be separated with spaces or dash
SQL> --What is inserted is both date and time
SQL> INSERT  INTO  date_test VALUES ('01 feb 11');

1 row created.

SQL> INSERT  INTO  date_test VALUES  ('01-feb-11');

1 row created.

SQL> --This will be inserted as year 2045
SQL> INSERT  INTO  date_test VALUES  ('01-feb-45');

1 row created.

SQL> INSERT  INTO  date_test VALUES  ('01-feb-1945');

1 row created.

SQL> --Insert the current date and time
SQL> INSERT  INTO  date_test VALUES  (SYSDATE);

1 row created.

SQL> --The time is not visible but it is in there
SQL> SELECT * FROM date_test;

COL
---------
01-FEB-11
01-FEB-11
01-FEB-45
01-FEB-45
12-DEC-11
```

```
--Use the to_char function to display in some other format. It will display each date in the (COL)
--column according to the codes .
SELECT TO_CHAR (col,'YYYY/MM/DD') FROM date_test;
SELECT TO_CHAR (col,'YY MM DY') FROM date_test;

--Note that (mi) is for minutes.
SELECT TO_CHAR (col,'YYYY/MM/DD hh24:mi:ss') FROM date_test;
```

```
SQL> --Use the to_char function to display in some other format
SQL> --It will display each date in the (COL) column according to the
SQL> --codes
SQL> SELECT TO_CHAR (col,'YYYY/MM/DD') FROM date_test;

TO_CHAR(CO
----------
2011/02/01
2011/02/01
2045/02/01
1945/02/01
2011/12/12

SQL> SELECT TO_CHAR (col,'YY MM DY') FROM date_test;

TO_CHAR(C
---------
11 02 TUE
11 02 TUE
45 02 WED
45 02 THU
11 12 MON

SQL> --Note that (mi) is for minutes.
SQL> SELECT TO_CHAR (col,'YYYY/MM/DD hh24:mi:ss') FROM date_test;

TO_CHAR(COL,'YYYY/M
-------------------
2011/02/01 00:00:00
2011/02/01 00:00:00
2045/02/01 00:00:00
1945/02/01 00:00:00
2011/12/12 10:36:56
```

```
--inserting a non-default format use to_date
INSERT  INTO  date_test VALUES
(TO_DATE('1999/02/04','YYYY/MM/DD'));

INSERT  INTO  date_test VALUES  (TO_DATE('99/02/04','YY/DD/MM'));

INSERT  INTO  date_test VALUES  (TO_DATE('99/02/04
23:23:20','YY/DD/MM HH24:MI:SS'));
```

```
SQL> --inserting a non-default format use to_date
SQL> INSERT  INTO  date_test VALUES  (TO_DATE('1999/02/04','YYYY/MM/DD'))

1 row created.

SQL> INSERT  INTO  date_test VALUES  (TO_DATE('99/02/04','YY/DD/MM'));

1 row created.



SQL> INSERT  INTO  date_test VALUES  (TO_DATE('99/02/04 23:23:20','YY/DD/MM HH2
:MI:SS'));

1 row created.
```

```
SELECT TO_CHAR (col,'YYYY/MM/DD hh24:mi:ss') FROM date_test;
```

```
SQL> SELECT TO_CHAR (col,'YYYY/MM/DD hh24:mi:ss') FROM date_test;

TO_CHAR(COL,'YYYY/M
-------------------
2011/02/01 00:00:00
2011/02/01 00:00:00
2045/02/01 00:00:00
1945/02/01 00:00:00
2011/12/12 10:36:56
1999/02/04 00:00:00
2099/04/02 00:00:00
2099/04/02 23:23:20

8 rows selected.
```

Example 4.3b (Change date format)

- Change the way date is dealt with for the session only.

```
--Change the way date is dealt with for the session only. These changes occur to the
--NLS_SESSION_PARAMETERS table. This becomes the new default format that needs to be
followed
--when inserting dates. Dates will also be displayed in this format.
ALTER SESSION SET nls_date_format = 'MON-DD-YY HH24:MI:SS';
SELECT sysdate FROM dual;

--INVALID: This does not fit the new format
INSERT  INTO  date_test VALUES ('01 feb 99');

--This does fit the new format.
INSERT INTO date_test VALUES ('FEB-01-99');
```

```
SQL> -- Change the way date is dealt with for the session only
SQL> -- These changes occur to the NLS_SESSION_PARAMETERS table
SQL> --This becomes the new default format that needs to be followed when
SQL> --inserting dates. Dates will also be displayed in this format
SQL> ALTER SESSION SET nls_date_format = 'MON-DD-YY HH24:MI:SS';

Session altered.

SQL> SELECT sysdate FROM dual;

SYSDATE
------------------
DEC-12-11 10:40:18

SQL>
SQL> -- This does not fit the new format
SQL> INSERT  INTO  date_test VALUES ('01 feb 99');
INSERT  INTO  date_test VALUES ('01 feb 99')
                               *
ERROR at line 1:
ORA-01843: not a valid month


SQL>
SQL> -- This does fit the new format
SQL> INSERT INTO date_test VALUES ('FEB-01-99');

1 row created.
```

```
--Change the way date is dealt with for the session
ALTER SESSION SET nls_date_format = 'dy mm-YY';
SELECT sysdate FROM dual;

--INVALID: What day of the month are we inserting?
INSERT INTO date_test VALUES ('mon:02-99');
```

```
SQL> -- Change the way date is dealt with for the session
SQL> ALTER SESSION SET nls_date_format = 'dy mm-YY';

Session altered.

SQL> SELECT sysdate FROM dual;

SYSDATE
---------
mon 12-11

SQL>
SQL> -- Problem because what day of the month are we inserting because it
SQL> --doesn't know which day of the month we are inserting
SQL> INSERT INTO date_test VALUES ('mon:02-99');
INSERT INTO date_test VALUES ('mon:02-99')
                              *
ERROR at line 1:
ORA-01835: day of week conflicts with Julian date
```

✓ *CHECK 4A*

1. Insert a record into the Person table. The date should be in the format mm/yyyy/dd hh24:mi:ss

2. Display the records in the Person table. The date should be displayed in the format yyyy/dd/mm

"The advantage of a bad memory is that one enjoys several times the same good things for the first time. "

4.4 Sequences

A sequence generates sequential integers that can serve as primary keys for tables. Sequences aren't assigned to a specific column or table. They are independent objects and, therefore, different users can use the same sequence to generate values that are inserted into several different tables. In other words, the same sequence could be used to generate order numbers and customer numbers. This use results in gaps in the sequence values appearing in each table

The INCREMENT BY clause specifies the interval between two sequential values. For checks and invoices, this interval is usually 1.

The START WITH clause establishes the starting value for the sequence.

The MINVALUE and MAXVALUE clauses establish a minimum or maximum value for the sequence. If the sequence is incre-mented with a positive value, using the MINVALUE clause doesn't make sense.

The CYCLE and NOCYCLE options determine whether Oracle should begin reis-suing values from the sequence after reaching the minimum or maximum value.

Example 4.4a (Create sequence)

--Creating a sequence

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;
DROP SEQUENCE patient_patient_id_seq;
DROP SEQUENCE disease_disease_id_seq;

CREATE  TABLE Patient
```

```
(
      Patient_id  NUMBER PRIMARY KEY,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);

CREATE  TABLE Disease
(
      diseaseid   NUMBER PRIMARY KEY,
      disease_desc  VARCHAR2(20)
);

--The sequence starts with 1 and then increments by 1.
--Note that the sequence is not associated with any table. It will be used in the insert
--statement and can actually be used with many tables
CREATE SEQUENCE patient_patient_id_seq START WITH 1;

--A sequence is created that starts with 10 and is incremented by 4
CREATE SEQUENCE disease_disease_id_seq INCREMENTBY 4 START WITH 10;
```

```
SQL>
SQL> CREATE  TABLE Patient
  2  (
  3     Patient_id      NUMBER PRIMARY KEY,
  4     Height          NUMBER,
  5     Fname           VARCHAR2(20),
  6     Lname           VARCHAR2(20)
  7  );

Table created.

SQL>
SQL> CREATE  TABLE Disease
  2  (
  3     diseaseid          NUMBER PRIMARY KEY,
  4     disease_desc   VARCHAR2(20)
  5  );

Table created.

SQL>
SQL> --A sequence is created that starts with 1 and then increments by 1
SQL> --Note that the sequence is not associated with any table. It will be use
in the insert
SQL> --statement and can actually be used with many tables
SQL> CREATE SEQUENCE patient_patient_id_seq START WITH 1;

Sequence created.

SQL> --A sequence is created that starts with 10 and is incremented by 4
SQL> CREATE SEQUENCE disease_disease_id_seq INCREMENT BY 4 START WITH 10;

Sequence created.
```

Example 4.4b (Get access to next sequence)

You can access sequence values by using the two pseudo-columns NEXTVAL and CURRVAL. The pseudo-column NEXTVAL is used to generate the sequence value. After a value is generated, it's stored in the CURRVAL (CURRENT VALUE) pseudo-column so that you can reference it again.

```
--Inserts the next number in the sequence.
INSERT INTO patient VALUES
(patient_patient_id_seq.nextval,10,'john','james');

/* Can use the DUAL table to find out what the next value is. When nextval is used, it
--automatically increments the sequence. There will be more discussion on the DUAL table. */
SELECT patient_patient_id_seq.nextval FROM  DUAL;

INSERT INTO patient VALUES
(patient_patient_id_seq.nextval,40,'jimm','jones');

SELECT * FROM patient;
```

```
SQL> --inserts the next number in the sequence
SQL> INSERT INTO patient VALUES (patient_patient_id_seq.nextval,10,'john','jam
');

1 row created.

SQL> --can use the DUAL table to find out what the next value is. When nextval
s used, it
SQL> --automatically increments the sequence. There will be more discussion on
he DUAL table
SQL> SELECT patient_patient_id_seq.nextval FROM  DUAL;

   NEXTVAL
----------
         3

SQL> INSERT INTO patient VALUES (patient_patient_id_seq.nextval,40,'jimm','jon
');

1 row created.

SQL> SELECT * FROM patient;

PATIENT_ID     HEIGHT FNAME                LNAME
---------- ---------- -------------------- --------------------
         2         10 john                 james
         4         40 jimm                 jones
```

Example 4.4c (User_sequences)

--Examining the user_sequences table

```
--This is a system table that contains information about all the sequences that the user has
--created.
DESC USER_SEQUENCES;
SELECT sequence_name FROM user_sequences;
```

```
SQL> DESC USER_SEQUENCES;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------

 SEQUENCE_NAME                             NOT NULL VARCHAR2(30)
 MIN_VALUE                                          NUMBER
 MAX_VALUE                                          NUMBER
 INCREMENT_BY                              NOT NULL NUMBER
 CYCLE_FLAG                                         VARCHAR2(1)
 ORDER_FLAG                                         VARCHAR2(1)
 CACHE_SIZE                                NOT NULL NUMBER
 LAST_NUMBER                               NOT NULL NUMBER

SQL> SELECT sequence_name FROM user_sequences;

SEQUENCE_NAME
------------------------------
DISEASE_DISEASE_ID_SEQ
PATIENT_PATIENT_ID_SEQ
```

Example 4.4d (Alter sequences)

You can change settings for a sequence by using the ALTER SEQUENCE command. However, any changes are applied only to values generated after the modifications are made. The only restrictions that apply to changing the sequence settings are as follows: The START WITH clause can't be changed because the sequence would have to be dropped and re-created to make this change. The changes can't make previously issued sequence values invalid.

```
--Can use the alter command to modify the increment value
ALTER SEQUENCE patient_patient_id_seq INCREMENT BY 2;

--Confirm the change. There will be more discussion on the DUAL table
SELECT patient_patient_id_seq.currval FROM  DUAL;
SELECT patient_patient_id_seq.nextval FROM  DUAL;
```

```
SQL> --Can use the alter command to modify the increment value
SQL> ALTER SEQUENCE patient_patient_id_seq INCREMENT BY 2;

Sequence altered.

SQL> --Confirm the change. There will be more discussion on the DUAL table
SQL> SELECT patient_patient_id_seq.currval FROM  DUAL;

   CURRVAL
----------
         4

SQL> SELECT patient_patient_id_seq.nextval FROM  DUAL;

   NEXTVAL
----------
         6
```

```
INSERT INTO disease VALUES (disease_disease_id_seq.nextval,'yellow
fever');

SELECT * FROM disease;

SELECT disease_disease_id_seq.currval FROM  DUAL;
```

```
SQL> INSERT INTO disease VALUES (disease_disease_id_seq.nextval,'yellow fever'

1 row created.

SQL> SELECT * FROM disease;

 DISEASEID DISEASE_DESC
---------- --------------------
        14 malaria
        22 yellow fever

SQL> SELECT disease_disease_id_seq.currval FROM  DUAL;

   CURRVAL
----------
        22
```

```
--This identifies the current value without incrementing the seqence.
SELECT patient_patient_id_seq.currval FROM  DUAL;

INSERT INTO disease VALUES (disease_disease_id_seq.nextval,'malaria');

SELECT disease_disease_id_seq.nextval FROM  DUAL;
```

```
SQL> --This identifies the current value without incrementing
SQL> SELECT patient_patient_id_seq.currval FROM  DUAL;

   CURRVAL
----------
         4

SQL>
SQL> INSERT INTO disease VALUES (disease_disease_id_seq.nextval,'malaria');

1 row created.

SQL> SELECT disease_disease_id_seq.nextval FROM  DUAL;

   NEXTVAL
----------
        18
```

Example 4.4e (Drop sequences)

```
DROP SEQUENCE patient_patient_id_seq;
```

✓ *CHECK 4B*

1. Create a sequence for the primary key of the Person table.
2. Insert a record into the Person table using the newly created sequence.

“A computer lets you make more mistakes faster than any invention in human history - with the possible exceptions of handguns and tequila. “

Summary Examples

```
DROP table test;
CREATE  TABLE test
(
    cola VARCHAR(3),
    colb CHAR(2),
    colc  NUMBER(3,2),
    cold NUMBER(3),
    cole  DATE,
    colf  char
);

--Inserts have to be done in the same sequence as the columns appear in the create table statement if the
--columns are not specified.
--VARCHAR inserts the data without padding the data with zeroes.
--CHAR pads the data with spaces to reach the max size.
--NUMBER(3,2):  one digit for the whole number and two digits for the precision.
--The precision will be rounded if it is more.
--The default date format is dd-mon-yy or yyyy.
--The keyword NULL or single quotes with no spaces can be used to represent NULL (void of data).
INSERT INTO TEST VALUES ('ab','a',1.234,123,'01-feb-1996',NULL);

/* Can identify the columns that you want without concern for the order in the create table statement. Keep in
mind that when inserting, constraints cannot be violated, which means that if there is a primary key constraint,
it requires data and thus, has to be included in the column list.
When inserting other than default date format, the to_date function has to be used. Both the date and time are
inserted into a date column*/
INSERT INTO TEST (colA, colE) VALUES ('ab',to_date('01/01/1990','mm/dd/yyyy'))

/* When displaying the date in some other format use to_char. Keep in mind, for inserting, use to_date, for
displaying use to_char. mi is for minutes and ss is for seconds. They are not displayed by default. */
SELECT to_char(colE,'mm day yyyy mi:ss') FROM TEST;

--Create a sequence named some_seq. The system tabl, user_sequences, contains all info on sequences.
CREATE SEQUENCE some_seq INCREMENT BY 2 START WITH 2;

--The value for colD is the current value of the sequence. All other columns will contain a NULL
INSERT INTO TEST (colD) VALUES(some_seq.currval);

--Insert the nextval of sequence into the table.
INSERT INTO TEST (colD) VALUES (some_seq.nextval);

--Display the current value of sequence.
SELECT some_seq.currval FROM dual;

--System table, user_sequences, contains sequence info.
SELECT sequence_name FROM user_sequences;

--Drop the sequence
DROP SEQUENCE some_seq;
```