1Z0-061: DML

Item: 1 (Ref:1Z0-061.9.2.2)

The `PHYSICIAN` table contains these columns:

```
PHYSICIAN_ID NUMBER NOT NULL PK
LAST_NAME VARCHAR2(30) NOT NULL
FIRST_NAME VARCHAR2(25) NOT NULL
LICENSE_NO NUMBER(7) NOT NULL
HIRE_DATE DATE
```

When new physician records are added, the `PHYSICIAN_ID` is assigned a sequential value using the `PHY_NUM_SEQ` sequence. The state licensing board assigns license numbers with valid license numbers being from 1000000 to 9900000.

You want to create an `INSERT` statement that will prompt the user for each physician's name and license number and insert the physician's record into the `PHYSICIAN` table with a hire date of today. The statement should generate an error if an invalid license number is entered.

Which `INSERT` statement should you use?

○
```
INSERT INTO physician
VALUES (phy_num_seq.NEXTVAL, '&lname', '&fname', &lno, sysdate)
WHERE &lno BETWEEN 1000000 and 9900000;
```

○
```
INSERT INTO physician
VALUES (phy_num_seq.NEXTVAL, '&lname', '&fname', &lno BETWEEN 1000000 and 9900000, sysdate);
```

○
```
INSERT INTO
(SELECT physician_id, last_name, first_name, license_no, hire_date
FROM physician
WHERE license_no BETWEEN 1000000 and 9900000
WITH CHECK OPTION)
VALUES (phy_num_seq.VALUE, '&lname', '&fname', &lno, sysdate);
```

○
```
INSERT INTO
(SELECT physician_id, last_name, first_name, license_no, hire_date
FROM physician
WHERE license_no BETWEEN 1000000 and 9900000
WITH CHECK OPTION)
VALUES (phy_num_seq.NEXTVAL, &lname, &fname, &lno, sysdate);
```

○
```
INSERT INTO
(SELECT physician_id, last_name, first_name, license_no, hire_date
FROM physician
WITH CHECK OPTION
WHERE license_no BETWEEN 1000000 and 9900000)
VALUES (phy_num_seq.NEXTVAL, '&lname', '&fname', &lno, sysdate);
```

○
```
INSERT INTO
(SELECT physician_id, last_name, first_name, license_no, hire_date
FROM physician
WHERE license_no BETWEEN 1000000 and 9900000
WITH CHECK OPTION)
VALUES (&phy_num_seq, '&lname', '&fname', &lno, sysdate);
```

Answer:

```
INSERT INTO
(SELECT physician_id, last_name, first_name, license_no, hire_date
FROM physician
WITH CHECK OPTION
WHERE license_no BETWEEN 1000000 and 9900000)
VALUES (phy_num_seq.NEXTVAL, '&lname', '&fname', &lno, sysdate);
```

Explanation:

To perform the necessary insert, you should use the `INSERT` statement that uses a subquery including the `WITH CHECK OPTION` keyword to identify the table for the insert and uses `phy_num_seq.NEXTVAL` as the value to be inserted for `PHYSICIAN_ID`. When using a subquery for the table of a Data Manipulation Language (DML) statement, the `WITH CHECK OPTION` keyword can be used to ensure that the DML statement is not allowed if the change would generate rows that are not included in the subquery.

The `INSERT` statement that includes a `WHERE` clause is incorrect because a `WHERE` clause is not allowed with an `INSERT` statement.

The `INSERT` statement that uses the `BETWEEN` operator in the `VALUES` clause is incorrect because the `BETWEEN` operator cannot be used in a `VALUES` clause.

The `INSERT` statement that uses `phy_num_seq.VALUE` as the value inserted into the `PHYSICIAN_ID` column is incorrect and will cause an error. To generate the next sequence value from the `PHY_NUM_SEQ` sequence, you should use the `NEXTVAL` keyword.

The `INSERT` statement that does not include single quotation marks around the `&lname` and `&fname` substitution variables is incorrect because character and date substitution variables should be enclosed in single quotation marks.

The `INSERT` statement that uses `&phy_num_seq` as the value to be inserted for `PHYSICIAN_ID` will prompt the user for a value for `PHYSICIAN_ID`, rather than using the sequence as desired. Therefore, this option is incorrect.

Item: 2 (Ref:1Z0-061.9.2.3)

The `STUDENT` table contains these columns:

```
STU_ID NUMBER(9) NOT NULL
LAST_NAME VARCHAR2(30) NOT NULL
FIRST_NAME VARCHAR2(25) NOT NULL
DOB DATE
STU_TYPE_ID VARCHAR2(1) NOT NULL
ENROLL_DATE DATE
```

You create another table, named `PT_STUDENT`, with an identical structure. You want to insert all part-time students, who have a `STU_TYPE_ID` value of `P`, into the new table. You execute this `INSERT` statement:

```
INSERT INTO pt_student
(SELECT stu_id, last_name, first_name, dob, sysdate
FROM student
WHERE UPPER(stu_type_id) = 'P');
```

What is the result of executing this `INSERT` statement?

- ○ All part-time students are inserted into the `PT_STUDENT` table.
- ○ An error occurs because the `PT_STUDENT` table already exists.
- ○ An error occurs because you cannot use a subquery in an `INSERT` statement.
- ○ An error occurs because the `INSERT` statement does not contain a `VALUES` clause.
- ○ An error occurs because the `STU_TYPE_ID` column is not included in the subquery select list.
- ○ An error occurs because both the `STU_TYPE_ID` and `ENROLL_DATE` columns are not included in the subquery select list.

Answer:

An error occurs because the `STU_TYPE_ID` column is not included in the subquery select list.

Explanation:

When executing the given `INSERT` statement, an `ORA-00947: not enough values` error occurs because the `STU_TYPE_ID` column is not included in the subquery select list. When using a subquery to insert rows from one table into another table, the number and data types of the columns being inserted must match the number and data types of the columns returned by the subquery. In the given `INSERT` statement, no column list was included. This implies that all columns in the table will be inserted. The subquery in the statement, returns values for the `STU_ID`, `last_name`, and `DOB` columns, and uses `SYSDATE` for the `ENROLL_DATE` column. The `STU_TYPE_ID` column, however, is not included, and an error occurs.

All part-time students are not inserted into the `PT_STUDENT` table because this statement generates an error. If the select list of the subquery had included the `STU_TYPE_ID` column or a character constant had been included to give this column a value, all part-time students would have been inserted into the `PT_STUDENT` table.

The option stating that an error occurs because the `PT_STUDENT` table already exists is incorrect. In fact, to use a table in the `INTO` portion of a `SELECT` statement, the table must exist. You can however, use a subquery in a `CREATE TABLE` statement to create the table and insert records if needed.

The option stating that an error occurs because you cannot use a subquery in an `INSERT` statement is incorrect. Subqueries can be used both in the `INTO` portion of a `SELECT` statement and as a substitute for a `VALUES` clause in an `INSERT` statement.

The option stating that an error occurs because the `INSERT` statement does not contain a `VALUES` clause is incorrect. When including a subquery for the values to be inserted, the subquery replaces the `VALUES` clause.

The option stating that an error occurs because both the `STU_TYPE_ID` and `ENROLL_DATE` columns are not included in the subquery select list is incorrect because a valid date value, namely `SYSDATE`, was provided for the `ENROLL_DATE` column. Therefore, the `ENROLL_DATE` column is not a problem in this `INSERT` statement.

Item: 3 (Ref:1Z0-061.9.3.2)

Click the Exhibit(s) button to examine the data from the `po_header` and `po_detail` tables.

Examine the structures of the `po_header` and `po_detail` tables:

```
PO_HEADER
--------------------
PO_NUM      NUMBER  NOT NULL
PO_DATE     DATE  DEFAULT SYSDATE
PO_TOTAL    NUMBER(9,2)
SUPPLIER_ID NUMBER(9)
PO_TERMS    VARCHAR2(25)

PO_DETATIL
------------------
PO_NUM      NUMBER  NOT NULL
PO_LINE_ID  NUMBER  NOT NULL
PRODUCT_ID  NUMBER  NOT NULL,
QUANTITY    NUMBER(3)  NOT NULL,
UNIT_PRICE  NUMBER (5,2)  DEFAULT 0,
```

The primary key of the `po_header` table is `po_num`. The primary key of the `po_detail` table is the combination of `po_num` and `po_line_id`. A foreign key constraint is defined on the `po_num` column of the `po_detail` table that references the `po_header` table.

You want to update the purchase order total amount for a given purchase order. The `po_total` column in the `po_header` table should equal the sum of the extended amounts of the corresponding `po_detail` records. You want the user to be prompted for the purchase order number when the query is executed. When a purchase order is updated, the `po_date` column should be reset to the current date.

Which `UPDATE` statement should you execute?

○
```
UPDATE po_header
SET po_total = (SELECT SUM(ext)
FROM (SELECT po_num, quantity * unit_price ext
FROM po_detail
WHERE po_num = &&ponum)),
SET po_date = sysdate
WHERE po_num = &&ponum;
```

○
```
UPDATE po_header
SET po_total = (SELECT SUM(quantity * unit_price)
FROM (SELECT po_num)
FROM po_detail
WHERE po_num = &&ponum)),
po_date = DEFAULT
WHERE po_num = &&ponum;
```

○
```
UPDATE po_header
SET po_total = (SELECT SUM(ext)
FROM (SELECT po_num, quantity * unit_price ext
FROM po_detail
WHERE po_num = &&ponum)),
UPDATE po_header
SET po_date = sysdate
WHERE po_num = &&ponum;
```

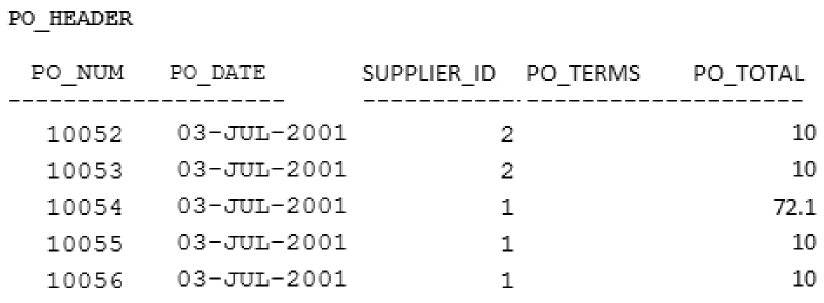
○
```
UPDATE po_header
SET po_total = (SELECT SUM(ext)
FROM (SELECT po_num, quantity * unit_price ext
FROM po_detail
WHERE po_num = &&ponum)),
po_date = DEFAULT
WHERE po_num = &&ponum;
```

○
```
UPDATE po_header
SET po_total = (SELECT po_num, SUM(ext)
FROM (SELECT po_num, quantity * unit_price ext
FROM po_detail
WHERE po_num = &&ponum)),
po_date = DEFAULT
WHERE po_num = &&ponum;
```

○
```
UPDATE po_header
SET po_total = (SELECT SUM(ext)
FROM (SELECT po_num, quantity * unit_price ext
FROM po_detail
WHERE po_num = &&ponum)),
po_date = NULL
WHERE po_num = &&ponum;
```

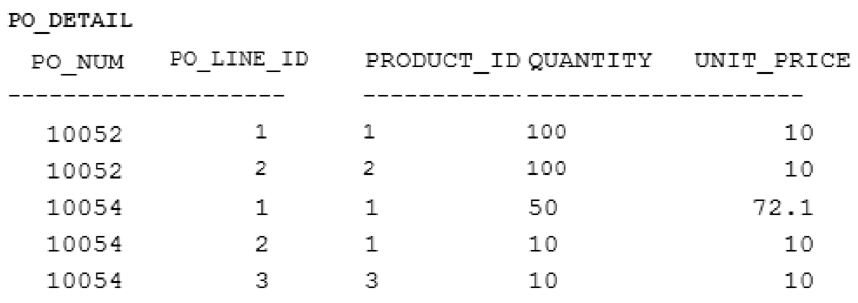
Answer:

```
UPDATE po_header
SET po_total = (SELECT SUM(ext)
FROM (SELECT po_num, quantity * unit_price ext
FROM po_detail
WHERE po_num = &&ponum)),
po_date = DEFAULT
WHERE po_num = &&ponum;
```



PO_HEADER

PO_NUM	PO_DATE	SUPPLIER_ID	PO_TERMS	PO_TOTAL
10052	03-JUL-2001	2		10
10053	03-JUL-2001	2		10
10054	03-JUL-2001	1		72.1
10055	03-JUL-2001	1		10
10056	03-JUL-2001	1		10



PO_DETAIL

PO_NUM	PO_LINE_ID	PRODUCT_ID	QUANTITY	UNIT_PRICE
10052	1	1	100	10
10052	2	2	100	10
10054	1	1	50	72.1
10054	2	1	10	10
10054	3	3	10	10

Explanation:

To perform the desired updates, you should execute the following `UPDATE` statement:

```
UPDATE po_header
SET po_total = (SELECT SUM(ext)
FROM (SELECT po_num, quantity * unit_price ext
FROM po_detail
WHERE po_num = &&ponum)),
po_date = DEFAULT
WHERE po_num = &&ponum;
```

Subqueries are always evaluated from innermost to outermost. First, the innermost query executes and returns the `po_num` and extended amount of each detail line of the specified purchase order. Then, the other subquery accepts this result and sums the extended amounts. The result is the sum of the extended amounts for each line item on the selected purchase order. The `po_total` column is updated with this value. The `po_date` column is updated using the `DEFAULT` keyword. When the `DEFAULT` keyword is used in an `UPDATE` or `INSERT` statement, the default value for the column being modified is used. In this scenario, the `po_date` column in the `po_header` table has a default value of `SYSDATE.` Therefore, the `po_date` is updated to the current date.

The `UPDATE` statement that includes more than one `SET` keyword is incorrect. The correct `UPDATE` statement syntax includes the `SET` keyword one time, followed by the columns to be updated separated by commas.

The `UPDATE` statement that includes `SUM(quantity * unit_price)` in the first subquery is incorrect. This subquery uses another subquery in its `FROM` clause, so only columns returned by the innermost query are available for use.

The `UPDATE` statement that nests another `UPDATE` statement within it is incorrect because `UPDATE` statements cannot be nested.

The `UPDATE` statement that returns both `po_num` and `SUM(ext)` in the first subquery is incorrect because this subquery result is compared using the `=` operator. Therefore, this query must return only one value.

The `UPDATE` statement that uses the `NULL` keyword to update the `po_date` column is incorrect. In this scenario, you wanted to update `po_date` to the current date. To do so, you could use `SYSDATE` or `DEFAULT`. Using the `NULL` keyword will update the `po_date` column to a `NULL` value instead.

When the `DEFAULT` keyword is used and no default value is defined for a column, the column is assigned a `NULL` value.

Item: 4 (Ref:1Z0-061.9.3.1)

Examine the structures of the DEPARTMENT and ASSET tables:

```
DEPARTMENT
-------------------------
DEPT_ID NUMBER(9) NOT NULL
DEPT_ABBR VARCHAR2(4)
DEPT_NAME VARCHAR2(25) NOT NULL
MGR_ID NUMBER

ASSET
-----------
ASSET_ID NUMBER(9) NOT NULL
ASSET_VALUE FLOAT
ASSET_DESCRIPTION VARCHAR2(25)
DEPT_ID NUMBER(9)
```

The dept_id column of the ASSET table has a foreign key constraint referencing the DEPARTMENT table. You attempt to update the ASSET table using this statement:

```
UPDATE asset
SET dept_id =
(SELECT dept_id
FROM department
WHERE dept_name =
(SELECT dept_name
FROM department
WHERE dept_abbr = 'FINC')),
asset_value = 10000
WHERE asset_id = 2;
```

Which two of the following statements must be true for this UPDATE statement to execute without generating an error? (Choose two.)

- ☐ An asset with an asset_id value of 2 must exist in the ASSET table.
- ☐ Only one row in the department table can have a dept_abbr value of FINC.
- ☐ One of the subqueries should be removed because subqueries cannot be nested.
- ☐ Both of the subqueries used in the UPDATE statement must return one and only one non-null value.
- ☐ Only one row in the department table can have the same dept_name value as the department with dept_abbr of FINC.

Answer:

Only one row in the department table can have a dept_abbr value of FINC.

Only one row in the department table can have the same dept_name value as the department with dept_abbr of FINC.

Explanation:

When executing the given UPDATE statement, each of the subqueries must return only one row. Because the equality (=) operator is used with each of the subqueries, each must return a single value or an error occurs. Therefore, only one row in the department table can have a dept_abbr value of FINC, and only one row in the department table can have the same dept_name value as the department with dept_abbr of FINC.

Although an asset with an asset_id value of 2 must exist for the intended update to be performed, it is not required for the given statement to execute without an error. The statement will execute successfully, but will perform no updates.

The option stating that one of the subqueries should be removed because subqueries cannot be nested is incorrect because nested subqueries are allowed. Subqueries can be nested as many times as needed to perform a task.

The option stating that both of the subqueries used in the UPDATE statement must return one and only one non-null value is also incorrect. An error does not occur if a subquery returns no values. However, your update result might not be as expected.

Item: 5 (Ref:1Z0-061.9.3.3)

The `product` table contains these columns:

```
PRODUCT_ID NUMBER NOT NULL
PRODUCT_NAME VARCHAR2(25)
SUPPLIER_ID NUMBER
LIST_PRICE NUMBER(7,2)
COST NUMBER(7,2)
```

You want to execute one DML statement to increase the cost of all products with a product name of Widget Connector by 10 percent and change the cost of all products with a description of Widget C - Round to equal the new cost of Widget Connector. Currently, all models of Widget Connectors have the same cost value.

Which statement should you execute?

- ◯
```
UPDATE product SET cost =
(SELECT DISTINCT cost * 1.10
FROM product
WHERE product_name = 'Widget Connector')
WHERE product_name IN('Widget C - Round', 'Widget Connector');
```
- ◯
```
UPDATE product SET cost =
(SELECT DISTINCT cost * .10
FROM product
WHERE product_name = 'Widget Connector')
WHERE product_name IN('Widget C - Round', 'Widget Connector');
```
- ◯
```
UPDATE product SET cost =
(SELECT cost * 1.10
FROM product
WHERE product_name = 'Widget Connector');
```
- ◯
```
UPDATE product SET cost =
(SELECT DISTINCT cost * 1.10
FROM product
WHERE product_name = 'Widget Connector'
OR product_name = 'Widget C - Round')
WHERE product_name = 'Widget Connector';
```
- ◯ You cannot perform these updates using one DML statement.

Answer:

```
UPDATE product SET cost =
(SELECT DISTINCT cost * 1.10
FROM product
WHERE product_name = 'Widget Connector')
WHERE product_name IN('Widget C - Round', 'Widget Connector');
```

Explanation:

You should execute the following statement:

```
UPDATE product SET cost =
(SELECT DISTINCT cost * 1.10
FROM product
WHERE product_name = 'Widget Connector')
WHERE product_name IN('Widget C - Round', 'Widget Connector');
```

The subquery retrieves the cost value of Widget Connectors increased by 10 percent. This value is then used as the new cost value of products with the description of `Widget Connector` and `Widget C - Round`. Because the question states that all Widget Connectors currently have the same cost and because the `DISTINCT` keyword is used in the subquery, the subquery returns only one row. This is required because a single-row operator is used with the subquery.

The statement that calculates the new cost as `cost * .10` is incorrect because it will only set the new cost values to 10 percent of the original value, not increase them by 10 percent.

The statement that does not include the `DISTINCT` keyword in the subquery will cause an error. The question implies that there is more than one product with a `product_name` of `Widget Connector`. Therefore, this query will return multiple values and cannot be used with a single-row operator (=).

The statement that includes an `OR` condition in the `WHERE` clause of the subquery is incorrect. The inner query will return increased cost values for products with either the name `Widget Connector` or the name `Widget C - Round`. If these products have different costs, the query returns more than one row and an error is generated.

Subqueries used in a comparison with a single-row operator (such as =, >, <, >=, <=, and <>) must return only one row.
Subqueries used in a comparison with a multiple-row operator (such as `IN`, `ANY`, and `ALL`) can return multiple rows.

Item: 6 (Ref:1Z0-061.9.3.4)

The `product` table contains these columns:

```
PRODUCT_ID NUMBER NOT NULL
PRODUCT_NAME VARCHAR2(25)
SUPPLIER_ID NUMBER
LIST_PRICE NUMBER(7,2)
COST NUMBER(7,2)
```

You need to increase the list price and cost of all products supplied by GlobeComm, Inc. by 5.5 percent. The `supplier_id` for GlobeComm is `105`.

Which statement should you use?

○
```
UPDATE product
SET list_price = list_price * 1.055
SET cost = cost * 1.055
WHERE supplier_id = 105;
```

○
```
UPDATE product
SET list_price = list_price * .055 AND
cost = cost * .055
WHERE supplier_id = 105;
```

○
```
UPDATE product
SET list_price = list_price * 1.055, cost = cost * 1.055
WHERE supplier_id = 105;
```

○
```
UPDATE product
SET list_price = list_price + (list_price * .055), cost = cost + (cost * .055)
WHERE supplier_id LIKE 'GlobeComm, Inc.';
```

Answer:

```
UPDATE product
SET list_price = list_price * 1.055, cost = cost * 1.055
WHERE supplier_id = 105;
```

Explanation:

You should use the following statement:

```
UPDATE product
SET list_price = list_price * 1.055, cost = cost * 1.055
WHERE supplier_id = 105;
```

In this scenario, you want to update the list price and cost by 5.5 percent for all products supplied by GlobeComm Corporation. This statement will correctly perform the needed updates. The `WHERE` clause will restrict those records updated to only those records with `supplier_id` equal to 105. The `SET` clause will update both the `list_price` and `cost` columns appropriately.

The `UPDATE` statement that includes two `SET` clauses and the statement that includes the AND operator in the `SET` clause are both incorrect. To update multiple columns in one `UPDATE` statement, the columns should be separated with commas and listed in one `SET` clause. Including more than one `SET` clause or including the `AND` operator in the `SET` clause will generate an error.

The statement that includes the `LIKE` operator in the `WHERE` clause is incorrect. `supplier_id` is a numeric value, and the `LIKE` operator is only valid with columns that have a character data type.

Item: 7 (Ref:1Z0-061.9.4.3)

Evaluate this statement:

```
DELETE FROM workorder;
```

What does this statement accomplish?

- ○ deletes the `workorder` column
- ○ discards only the structure of the `workorder` table
- ○ deletes all the rows from the `workorder` table
- ○ deletes all the values in the columns that do not have `NOT NULL` constraints
- ○ deletes all rows from the `workorder` table and permanently discards the table's structure
- ○ generates an error because the `FROM` keyword should not be included

Answer:

deletes all the rows from the `workorder` table

Explanation:

The given statement will delete all rows from the `workorder` table. The `DELETE` statement removes existing rows from a table, but does not affect the table's structure. If you omit the `WHERE` clause, all the rows in the table will be deleted.

To delete a column from a table, use the `ALTER TABLE` statement.

To discard the structure of the table, use the `DROP TABLE` statement. The `DELETE` statement does not alter the table structure in any way, but only affects the data.

This statement does not generate an error because the `FROM` keyword is allowed in the `DELETE` statement. However, the `FROM` keyword can be omitted, and the same result occurs.

When used without a `WHERE` clause, the `DELETE` statement deletes all rows from a table, not just those that do not have `NOT NULL` constraints. Therefore, this option is incorrect.

The `DELETE` statement does not affect the table's structure. Therefore, the option stating that it deletes all rows from the `workorder` table and permanently discards the table's structure is incorrect.

Item: 8 (Ref:1Z0-061.9.2.1)

The `product` table contains these columns:

```
PRODUCT_ID NUMBER NOT NULL
PRODUCT_NAME VARCHAR2(25)
SUPPLIER_ID NUMBER NOT NULL
LIST_PRICE NUMBER(7,2)
COST NUMBER(5,2)
QTY_IN_STOCK NUMBER(4)
LAST_ORDER_DT DATE DEFAULT SYSDATE NOT NULL
```

Which `INSERT` statement will execute successfully?

- ○
```
INSERT INTO product
VALUES (10,'Ladder-back Chair', 5, 59.99, 37.32 , 1000, 10-JAN-08);
```
- ○
```
INSERT INTO product
VALUES (10,'Ladder-back Chair', 5, 59.99, 37.32 , 2, DEFAULT);
```
- ○
```
INSERT INTO product(product_id, supplier_id, list_price, last_order_dt)
VALUES (10, 5, 65.99);
```
- ○
```
INSERT INTO product
VALUES (10,'Ladder-back Chair', NULL, NULL, NULL, NULL, DEFAULT);
```
- ○
```
INSERT INTO product
VALUES (10,'Ladder-back Chair', 5, 59.99, 37.32 , 10000, DEFAULT);
```

Answer:

```
INSERT INTO product
VALUES (10,'Ladder-back Chair', 5, 59.99, 37.32 , 2, DEFAULT);
```

Explanation:

The following `INSERT` statement will execute successfully:

```
INSERT INTO product
VALUES (10,'Ladder-back Chair', 5, 59.99, 37.32 , 2, DEFAULT);
```

This statement is correct because the number and data type of the items in the values list matches that specified in the column list and a value is provided for all `NOT NULL` columns. Substitution variables can be used inside an `INSERT` statement to create reusable scripts. Each time the statement is executed, the user is prompted for the values of the substitution variables.

The option that uses an explicit value of `10-JAN-08` for the `last_order_dt` in the `VALUES` clause is incorrect. Date and character values within a `VALUES` clause must be enclosed in single quotation marks.

The option that uses an explicit value of `10000` for `qty_in_stock` in the `VALUES` clause is incorrect because this value is larger than the column definition allows.

The option that includes four columns in the column list and only three values in the `VALUES` clause is incorrect. If columns are explicitly provided in a column list, then all columns listed must be included in the `VALUES` clause with an explicit value, `NULL`, or `DEFAULT`.

The option that specifies `NULL` in the `VALUES` clause for the `supplier_id`, `list_price`, `cost`, and `qty_in_stock` columns is incorrect. The `supplier_id` column has a `NOT NULL` constraint and attempting to insert a `NULL` value for this column will generate an error.

Item: 9 (Ref:1Z0-061.9.4.1)

Examine the structures of the CURR_ORDER and LINE_ITEM tables:

```
CURR_ORDER
-------------------------
ORDER_ID NUMBER(9)
CUSTOMER_ID NUMBER(9)
ORDER_DATE DATE
SHIP_DATE DATE

LINE_ITEM
------------------
LINE_ITEM_ID NUMBER(9)
ORDER_ID NUMBER(9)
PRODUCT_ID NUMBER(9)
QUANTITY NUMBER(5)
```

The ORDER_ID column in the LINE_ITEM table has a foreign key constraint to the CURR_ORDER table.

Which statement about these two tables is TRUE?

- ○ To insert a row into the CURR_ORDER table, you must insert a row into the LINE_ITEM table.
- ○ To delete a row from the CURR_ORDER table, you must first delete any child rows from the LINE_ITEM table.
- ○ To update a row in the CURR_ORDER table, the parent row must already exist in the LINE_ITEM table.
- ○ To remove the constraint from the LINE_ITEM table, you must delete all the corresponding rows in the CURR_ORDER table.
- ○ To delete a row from the LINE_ITEM table, you must delete the associated row in the CURR_ORDER table.
- ○ When a row is deleted from the LINE_ITEM table, the associated parent row in the CURR_ORDER table is also deleted.

Answer:

To delete a row from the CURR_ORDER table, you must first delete any child rows from the LINE_ITEM table.

Explanation:

Because the ORDER_ID column in the LINE_ITEM table has a foreign key constraint referencing the ORDER_ID column in the CURR_ORDER table, you must delete any child rows from the LINE_ITEM table before deleting the corresponding row from the CURR_ORDER table. In this relationship, defined by the foreign key constraint, the CURR_ORDER table is the parent table and the LINE_ITEM table is the child table. The foreign key constraint ensures that no child (LINE_ITEM) can be created unless it has a parent (CURR_ORDER) and that no parent (CURR_ORDER) can be deleted if it has one or more children (LINE_ITEM).

With this defined relationship, you would also receive an integrity constraint error if you attempted to insert a row into the LINE_ITEM table and a parent row did not exist in the CURR_ORDER table.

You can insert a row into the CURR_ORDER table that has no associated LINE_ITEM row because the CURR_ORDER table is the parent table.

The option stating that you can update a row in the CURR_ORDER table if the parent already exists in the LINE_ITEM table is incorrect. In this scenario, the CURR_ORDER table is the parent table and the LINE_ITEM table is the child table.

To remove the constraint from the LINE_ITEM table, no conditions must be met. Therefore, the option that states you must delete all records in the CURR_ORDER table is incorrect.

A parent is not required to have children records, so you can delete a row from the child table without necessarily deleting any rows from the parent table. When a row is deleted from the LINE_ITEM table, the associated parent row in the CURR_ORDER table is not deleted. The order might contain more than one line item, and therefore the parent must be retained. You can, however, specify the ON DELETECASCADE option when defining the foreign key constraint to provide for automatically deleting children rows when the parent is deleted.

Item: 10 (Ref:1Z0-061.9.5.1)

Which two statements would cause an implicit `COMMIT` to occur? (Choose two.)

- ☐ `GRANT`
- ☐ `SELECT`
- ☐ `RENAME`
- ☐ `COMMIT`
- ☐ `UPDATE`
- ☐ `ROLLBACK`

Answer:

`GRANT`
`RENAME`

Explanation:

Data Control Language (DCL) and Data Definition Language (DDL) statements cause an implicit commit when issued. DCL statements consist of commands such as `GRANT` and `REVOKE`, and are used to control access to the database and data. DDL statements are used to create database objects and consist of statements like `CREATE`, `DROP`, `ALTER`, and `RENAME`.

All of the other options are incorrect because they do not cause an implicit `COMMIT` to occur. The `SELECT` statement queries data from the database. Data Manipulation Language (DML) statements such as `UPDATE`, `DELETE`, `INSERT`, and `MERGE` do not cause an implicit commit when issued. Transaction Control Language (TCL) statements such as `COMMIT` and `ROLLBACK` do not cause an implicit commit either. The `COMMIT` statement causes an explicit commit of a transaction. The `ROLLBACK` statement rolls back any uncommitted transactions.

A transaction begins when a DML statement is issued. The transaction terminates when an explicit `COMMIT` or `ROLLBACK` is executed, a DDL or DCL statement is encountered, the user exits the session, or the session terminates abnormally, such as with a system crash or machine failure.

Item: 11 (Ref:1Z0-061.9.5.2)

The `product` table contains these columns:

```
PRODUCT_ID NUMBER PK
NAME VARCHAR2(30)
LIST_PRICE NUMBER(7,2)
COST NUMBER(7,2)
```

You logged on to the database to update the `product` table. After your session began, you issued these statements:

```
INSERT INTO product VALUES(4,'Ceiling Fan',59.99,32.45);
INSERT INTO product VALUES(5,'Ceiling Fan',69.99,37.20);
SAVEPOINT A;
UPDATE product SET cost = 0;
SAVEPOINT B;
DELETE FROM product WHERE UPPER(name) = 'CEILING FAN';
ALTER TABLE product ADD qoh NUMBER DEFAULT 10;
ROLLBACK TO B;
UPDATE product SET name = 'CEILING FAN KIT' WHERE product_id = 4;
```

Then, you exit the session.

Which of the DML statements in this script performed either an `INSERT`, `UPDATE`, or `DELETE` that affected at least one row?

- ◯ only the `INSERT` statements
- ◯ only the `INSERT` statements and the first `UPDATE` statement
- ◯ the `INSERT` statements, the first `UPDATE` statement, and the `DELETE` statement
- ◯ all of the DML operations
- ◯ none of the DML operations

Answer:

the `INSERT` statements, the first `UPDATE` statement, and the `DELETE` statement

Explanation:

In this example, the `INSERT` statements, the `UPDATE` statement, and the `DELETE` statement are committed by this script. When the `ALTER TABLE` statement is executed, an implicit commit occurs. This commits the updates performed in both `INSERT` statements, the first `UPDATE` statement, and the `DELETE` statement. This implicit commit releases all held locks, erases all savepoints, and writes the changes permanently to the database.

When the `ROLLBACK TO B` statement is issued, an error will occur stating that savepoint B was never established. Therefore, this rollback statement has no effect. Then, the final `UPDATE` statement is issued but never committed.

Another implicit commit may possibly occur when you exit the session, and if so, this implicit `COMMIT` commits the changes made by the last `UPDATE` statement. However, that work is not part of the script. This action will depend on the SQL*Plus settings regarding the treatment of pending transactions when you exit that particular tool.

All options indicating that fewer or more rows were actually updated are false.

Item: 12 (Ref:1Z0-061.9.4.2)

Click the **Exhibit(s)** button to examine the structures of the `product` and `supplier` tables. You want to delete any products supplied by suppliers located in Dallas that have an in-stock quantity less than a specified value.

Which statement should you use?

◯
```
DELETE FROM product
WHERE supplier_id =
(SELECT supplier_id
FROM supplier
WHERE UPPER(city) = 'DALLAS')
AND qty_in_stock < &qoh;
```

◯
```
DELETE FROM product
WHERE supplier_id IN
(SELECT supplier_id
FROM supplier
WHERE UPPER(city) = 'DALLAS'
AND qty_in_stock < &qoh);
```

◯
```
DELETE FROM supplier
WHERE supplier_id IN
(SELECT supplier_id
FROM supplier
WHERE UPPER(city) = 'DALLAS')
AND qty_in_stock < &qoh;
```

◯
```
DELETE FROM product
WHERE supplier_id IN
(SELECT supplier_id
FROM supplier
WHERE UPPER(city) = 'DALLAS')
AND qty_in_stock < &qoh;
```

◯
```
DELETE FROM product
WHERE supplier_id IN
(SELECT supplier_id
FROM supplier
WHERE UPPER(city) = 'DALLAS'
AND supplier_id IN
(SELECT supplier_id
FROM product
WHERE qty_in_stock > &qoh));
```

Answer:

```
DELETE FROM product
WHERE supplier_id IN
(SELECT supplier_id
FROM supplier
WHERE UPPER(city) = 'DALLAS')
AND qty_in_stock < &qoh;
```

PRODUCT

PRODUCT_ID	NUMBER	NOT NULL, Primary Key
PRODUCT_NAME	VARCHAR2(25)	
SUPPLIER_ID	NUMBER	Foreign key to SUPPLIER_ID of the SUPPLIER table
LIST_PRICE	NUMBER(7,2)	
COST	NUMBER(7,2)	
QTY_IN_STOCK	NUMBER	
QTY_ON_ORDER	NUMBER	
REORDER_LEVEL	NUMBER	
REORDER_QTY	NUMBER	

SUPPLIER

SUPPLIER_ID	NUMBER	NOT NULL, Primary Key
SUPPLIER_NAME	VARCHAR2(25)	
ADDRESS	VARCHAR2(30)	
CITY	VARCHAR2(25)	
REGION	VARCHAR2(10)	
POSTAL_CODE	VARCHAR2(11)	

Explanation:

You should use the following statement:

```
DELETE FROM product
WHERE supplier_id IN
(SELECT supplier_id
FROM supplier
WHERE UPPER(city) = 'DALLAS')
AND qty_in_stock < &qoh;
```

The inner query returns a list of all suppliers in Dallas and passes this list to the main query. The main query then deletes only those products whose `supplier_id` is in the list of Dallas suppliers and whose `qty_in_stock` is less than the value input by the user.

The statement that uses the equality operator (=) with the subquery in the `WHERE` clause is incorrect. The subquery returns all suppliers with a city of `DALLAS`. Because the subquery can return more than one row, it cannot be used with a single-row operator.

The statement that uses `WHERE UPPER(city) = 'DALLAS' AND qty_in_stock < &qoh)` as the condition for the subquery is incorrect. The `qty_in_stock` column resides in the `product` table, and only the `supplier` table is listed in the subquery's `FROM` clause.

The statement that deletes from the `supplier` table is incorrect. You wanted to delete records from the `product` table, not the `supplier` table.

The statement that includes nested subqueries is incorrect. Subqueries can be nested if needed, but this statement's innermost query presents a problem. The innermost query returns a list of suppliers that supply products for which you have the desired quantity on hand. It returns the results to the next level query, which returns a list of suppliers meeting both of the conditions. This would be all suppliers who are from Dallas and have products with the desired quantity. The `DELETE` query deletes all products for this list of suppliers. These suppliers, although they do have products that need to be deleted, may have other products that do not meet the delete criteria. Therefore, this option is incorrect.