Item: 1 (Ref:1Z0-061.2.2.2)

The `account` table contains these columns:

```
ACCOUNT_ID NUMBER(12)
NEW_BALANCE NUMBER(7,2)
PREV_BALANCE NUMBER(7,2)
FINANCE_CHARGE NUMBER(7,2)
```

With the least amount of effort, you want to display all of the rows in the `account` table.
Which query should you use?

- ○
```
SELECT *
FROM account;
```
- ○
```
SELECT all
FROM account;
```
- ○
```
SELECT any
FROM account;
```
- ○
```
SELECT account_id, new_balance, prev_balance, finance_charge
FROM account;
```

Answer:

```
SELECT *
FROM account;
```

Explanation:

You should use the following query to display all of the `account` table rows:

```
SELECT *
FROM account;
```

The asterisk (*) in the `SELECT` clause selects all columns from a table, view, materialized view, or snapshot. Using the asterisk (*) in the `SELECT` list simplifies the writing of such a query because the column names do not have to be individually included.

You should not use the query that includes each column name in the `SELECT` list. Although this query will achieve the desired results, it requires more effort than using an asterisk (*) in the `SELECT` list.

Both of the other options are incorrect because these statements fail. `ALL` is a keyword used in a `FORUPDATE` clause or in a `WHERE` clause comparison condition. When used in a `SELECT` clause, the statement fails. `ANY` is an operator used in a `WHERE` clause comparison condition. When used in a `SELECT` clause, the statement fails.

Item: 2 (Ref:1Z0-061.2.2.5)

Which statement, when executed, displays a zero if the `prev_balance` value is null and the `new_balance` value is zero?

- `SELECT NVL(.009 * prev_balance, 0) + new_balance "Current Balance" FROM account;`
- `SELECT NULL(.009 * prev_balance, 0) + new_balance "Current Balance" FROM account;`
- `SELECT IS NULL(.009 * prev_balance, 0) + new_balance "Current Balance" FROM account;`
- `SELECT TO_NUMBER(.009 * prev_balance) + new_balance "Current Balance" FROM account;`

Answer:

```
SELECT NVL(.009 * prev_balance, 0) + new_balance "Current Balance"
FROM account;
```

Explanation:

The following statement displays a zero if the `prev_balance` value is null and the `new_balance` value is zero:

```
SELECT NVL(.009 * prev_balance, 0) + new_balance "Current Balance"
FROM account;
```

If a column value in an arithmetic expression is null, the expression evaluates to null. If the value of the `prev_balance` column is null, the expression `.009 * prev_balance` returns a null value. When the null value is then used in an arithmetic expression, the value returned is `UNKNOWN`. When the value is `UNKNOWN`, no value is displayed.

The first expression, `.009 * prev_balance`, is evaluated first because multiplication takes precedence over addition. If the `prev_balance` value is null, it will be replaced with a 0, which is then used as part of an arithmetic calculation. The `NVL` function is used to replace a null value with a value. The first expression represents the column queried, and the second expression represents the string you want displayed if a null value is retrieved. The correct syntax of the `NVL` function is:

```
NVL(expression1, expression2)
```

The statement using `NULL(.009 * prev_balance, 0)` fails because `NULL` cannot be used by itself. It is not a function.

The statement using `IS NULL(.009 * prev_balance, 0)` fails because of the invalid use of the `IS NULL` comparison operator. The `IS NULL` operator should be used in the `WHERE` clause of a SQL statement to test for null values.

The statement containing `TO_NUMBER(.009 * prev_balance, 0)` fails because of the invalid use of the `TO_NUMBER` function. The `TO_NUMBER` function is used to convert a character data type value to a value of `NUMBER` data type or a specific format. This functionality is not needed in this example.

Item: 3 (Ref:1Z0-061.2.2.1)

The `account` table contains these columns:

```
ACCOUNT_ID NUMBER(12)
NEW_PURCHASES NUMBER(7,2)
PREV_BALANCE NUMBER(7,2)
FINANCE_CHARGE NUMBER(7,2)
PAYMENTS NUMBER(7,2)
```

You must print a report that contains the account number and the current balance for a particular customer. The current balance consists of the sum of an account's previous balance, new purchases, and finance charge. You must calculate the finance charge based on a rate of 1.5 percent of the previous balance. Payments must be deducted from this amount. The customer's account number is 543842.

Which `SELECT` statement should you use?

- ○
```
SELECT new_balance + finance_charge - payments
FROM account
WHERE account_id = 543842;
```
- ○
```
SELECT account_id, new_purchases + prev_balance * 1.015 - payments
FROM account
WHERE account_id = 543842;
```
- ○
```
SELECT account_id, new_purchases + (prev_balance * .015) - payments
FROM account
WHERE account_id = 543842;
```
- ○
```
SELECT account_id, new_purchases + (prev_balance * 1.015) + finance_charge -
payments
FROM account
WHERE account_id = 543842;
```

Answer:

```
SELECT account_id, new_purchases + prev_balance * 1.015 - payments

FROM account
WHERE account_id = 543842;
```

Explanation:

You should use the following `SELECT` statement:

```
SELECT account_id, new_purchases + prev_balance * 1.015 - payments
FROM account
WHERE account_id = 543842;
```

To calculate the new balance on an account, the finance charge is calculated by multiplying the previous balance by .015. To include the previous balance amount, 1.015 is used instead of .015 (`prev_balance * 1.015`). The result equals the previous balance plus the finance charge, which is .015 percent of the previous balance. After adding new purchases and subtracting the payments, the current balance calculation is complete.

Although parentheses indicate a higher precedence in arithmetic calculations in a SQL statement, they are not

needed in this scenario. Because multiplication has precedence over addition and subtraction and the parentheses surround the multiplication calculation, the parentheses have no impact.

The choice that starts with `SELECT account_id, new_purchases + (prev_balance * 1.015) + finance_charge - payments` executes, but does not return the desired results. Instead of using the `finance_charge` column, the finance charge value must be calculated based on a rate of .015.

The choice that starts with `SELECT new_balance + finance_charge - payments` returns undesired results because the required current balance calculation is not used.

The choice that starts with `SELECT` clause `SELECT account_id, new_purchases + (prev_balance * .015) - payments` calculates the new balance improperly by neglecting to add the previous balance into the calculation.

Item: 4 (Ref:1Z0-061.2.1.1)

You query the database with this SQL statement:

```
SELECT id_number, NVL(100 / quantity, 0)
FROM product;
```

Which SQL `SELECT` statement capabilities are performed by this query?

- ◯ selection only
- ◯ projection only
- ◯ selection and projection only
- ◯ projection, selection, and joining

Answer:

projection only

Explanation:

This query performs the projection capability of a `SELECT` statement, because only specific columns are returned. Projection is performed when the `SELECT` clause contains a column or column list. Using an asterisk (*) in the `SELECT` clause would also be an example of projection, because you would be choosing all columns.

All of the other options are incorrect. This query does not perform a selection because no restricting criteria are used. Restricting criteria would be placed in a `WHERE` clause, but this example does not contain a `WHERE` clause. For example, you could include `WHERE manufacturer_id = 'NF10032'` in the query to limit the display results to only those products with a `manufacturer_id` value of NF10032.

This query does not perform a join because only one table is queried. The join capability is used when two or more tables are joined, or linked, in a `WHERE` clause and data is queried.

Item: 5 (Ref:1Z0-061.2.2.3)

The `teacher` table contains these columns:

```
ID NUMBER(9) Primary Key
LAST_NAME VARCHAR2(25)
FIRST_NAME VARCHAR2(25)
SUBJECT_ID NUMBER(9)
```

Which query should you use to display only the full name of each teacher along with the identification number of the subject each teacher is responsible for teaching?

- ○
```
SELECT *
FROM teacher;
```
- ○
```
SELECT last_name, subject_id
FROM teacher;
```
- ○
```
SELECT last_name, first_name, id
FROM teacher;
```
- ○
```
SELECT last_name, first_name, subject_id
FROM teacher;
```

Answer:

```
SELECT last_name, first_name, subject_id
FROM teacher;
```

Explanation:

You should use the following query to display only the full name of each teacher along with the identification number of the subject each teacher is responsible for teaching:

```
SELECT last_name, first_name, subject_id
FROM teacher;
```

To restrict the columns displayed to `last_name`, `first_name`, and `subject_id`, you should place only these columns in the `SELECT` list. You should place the columns you want to display in the order you want them displayed and separate them with commas.

You should not use the statement that uses an asterisk (*) in the `SELECT` list because this statement returns all of the columns in the `teacher` table. It returns all of the information requested, the full name as well as the subject identification, but it also includes the identification number of the teacher, which is not a part of the desired results.

You should not use the statement that includes only the `last_name` and `subject_id` columns in the `SELECT` list because this statement does not return the full name, as required in this scenario. The full name consists of the `last_name` and the `first_name` columns.

You should not use the statement that includes the `last_name`, `first_name`, and `id` columns in the `SELECT` list because this statement does not include the subject identification. Instead, it includes the teacher identification number, `id`, which is not a part of the desired results.

Item: 6 (Ref:1Z0-061.2.2.6)

Examine the structure of the LINE_ITEM table shown in the exhibit:

LINE_ITEM

LINE_ITEM_ID	NUMBER(9)	NOT NULL, Primary Key
ORDER_ID	NUMBER(9)	NOT NULL, Primary Key, Foreign Key to ORDER_ID column of the CURR_ORDER table
PRODUCT_ID	NUMBER(9)	NOT NULL, Foreign Key to PRODUCT_ID column of the PRODUCT table
QUANTITY	NUMBER(9)	

You query the database with this SQL statement:

```
SELECT order_id||'-'||line_item_id||' '||product_id||' '||quantity "Purchase"
FROM line_item;
```

Which component of the SELECT statement is a literal?

- ○ '-'
- ○ ||
- ○ quantity
- ○ "Purchase"

Answer:

'-'

Explanation:

The literal component in this statement is '-'. A literal value is a value that is specified explicitly. When using the concatenation operator (||) in a SELECT list, any literal date, expression, number, or character value must be enclosed in single quotes. This statement contains three literal values ('-', ' ', and ' '), which is one hyphen and two individual spaces.

The concatenation operator (||) is not a literal. The concatenation operator is used to concatenate, or combine, data. By placing the concatenation operator between columns, expressions, spaces, or literal values, the items are combined and displayed as one concatenated value in the query results. The syntax of the concatenation operator is:

```
(column1|expression1)||(column2|expression2)[|| (column3|expression3)]...
```

"Purchase" is not a literal, but is a column alias for the concatenated string.

The quantity, order_id, line_item_id, and product_id components are columns, not literals.

Item: 7 (Ref:1Z0-061.2.2.7)

The STUDENT table contains the following columns:

```
LAST_NAME VARCHAR2(25)
FIRST_NAME VARCHAR2(25)
EMAIL VARCHAR2(50)
```

You are writing a SELECT statement to retrieve the names of students that do NOT have an e-mail address.

```
SELECT last_name||', '||first_name "Student Name"
FROM student
```

Which WHERE clause should you use to complete this statement?

- ◯ `WHERE email = NULL;`
- ◯ `WHERE email != NULL;`
- ◯ `WHERE email IS NULL;`
- ◯ `WHERE email IS NOT NULL;`

Answer:

`WHERE email IS NULL;`

Explanation:

You should use the WHERE clause that uses the IS NULL comparison operator. When testing for null values in SQL, the IS NULL comparison operator should be used. This operator returns a Boolean value of TRUE when a null value is found, and returns FALSE when a value exists. The rows returned by this query will consist of students with no e-mail address.

You should not use the WHERE clause that uses the NOT operator with the IS NULL operator. When using the NOT operator in combination with the IS NULL operator, a Boolean value of TRUE is returned when the condition is false. This statement will execute successfully, but will not return the desired results. The rows returned using this WHERE clause would be students with an e-mail address.

You should not use the WHERE clauses that use the = or != operators. An equality operator (=) or other comparison operator, such as !=, should not be used when testing for null values because a null value cannot be equal or unequal to another value. Comparisons between nulls and other values do not return a TRUE or FALSE value, but instead return a value of UNKNOWN.

If EMAIL contains:	Clause condition	Evaluates to:
value	email IS NULL	FALSE
value	email IS NOT NULL	TRUE
null	email IS NULL	TRUE
null	email IS NOT NULL	FALSE
value or null	email = NULL	UNKNOWN
value or null	email != NULL	UNKNOWN

Item: 8 (Ref:1Z0-061.2.2.4)

The `account` table contains these columns:

```
ACCOUNT_ID NUMBER(12)
NEW_BALANCE NUMBER(7,2)
PREV_BALANCE NUMBER(7,2)
FINANCE_CHARGE NUMBER(7,2)
```

You must create statements to be mailed to all account holders. Each customer's statement must include the account holder's previous balance and finance charge in this format:

```
Previous Balance: 5000 Finance Charge: 45
```

Which `SELECT` statement will produce these results?

- ◯ `SELECT Previous Balance: ||prev_balance|| Finance Charge: ||prev_balance * .009 FROM account;`
- ◯ `SELECT 'Previous Balance:' ||prev_balance|| 'Finance Charge:' ||prev_balance * .009 FROM account;`
- ◯ `SELECT 'Previous Balance: '||prev_balance||' Finance Charge: '||prev_balance * .009 FROM account;`
- ◯ `SELECT "Previous Balance: "||prev_balance||" Finance Charge: "||prev_balance * .009 FROM account;`

Answer:

```
SELECT 'Previous Balance: '||prev_balance||' Finance Charge: '||prev_balance
* .009
FROM account;
```

Explanation:

The following `SELECT` statement will produce the desired results:

```
SELECT 'Previous Balance: '||prev_balance||' Finance Charge: '||prev_balance * .009
FROM account;
```

The concatenation operator (`||`) is used to concatenate, or combine, data. By placing the concatenation operator between columns, expressions, spaces, or literal values, the items are combined and displayed as one concatenated value in the query results. The syntax of the concatenation operator is:

```
(column1|expression1)||(column2|expression2)[||(column3|expression3)]...
```

Literal values must be enclosed in single quotes. To add space between a column value and text, place the desired number of blank spaces inside the single quotation marks, such as with `prev_balance||' Finance Charge '`. In this example a column is combined, or concatenated, to the text `'Finance Charge:'` with spaces separating them and a single space added to the end of the text. This returns the data in the desired format.

Because text cannot be combined with a column value using the concatenation operator unless single quotes surround the text, these `SELECT` statements will result in an error when executed:

```
SELECT Previous Balance ||prev_balance|| Finance Charge ||prev_balance * .009
FROM account;

SELECT "Previous Balance "||prev_balance||" Finance Charge "||prev_balance * .009
FROM account;
```

The following `SELECT` statement will not result in an error, but will not return the desired results because the spaces that are required between the text and column values displayed do not exist in the query results:

```
SELECT 'Previous Balance' ||prev_balance|| 'Finance Charge' ||prev_balance * .009
FROM account;
```