1Z0-061: DDL

Item: 1 (Ref:1Z0-061.10.1.1)

Which statement about a table is true?

- ○ A table can have up to 10,000 columns.
- ○ The size of a table does not need to be specified.
- ○ A table cannot be created while users are using the database.
- ○ The structure of a table cannot be modified while the table is online.

Answer:

The size of a table does not need to be specified.

Explanation:

When creating a table, it is not necessary to specify the size of the table. The sizes of database objects are ultimately defined by the amount of space allocated to the database as a whole. You can use the `STORAGE` clause and block utilization parameters to specify storage and usage characteristics, but these are not required. The Oracle Server will use default values when these values are not specified.

Each table can have up to 1,000 columns and each column in a table must adhere to the standard database object-naming conventions. Tables can be created at any time, as long as the tablespace that will contain the table is online.

Whether or not users are accessing the database has no effect on the creation of a table.

You can modify the structure of a table by altering its storage parameters or modifying the columns in the table while the table is online. You can alter a table's structure even while its tablespace is offline.

Item: 2 (Ref:1Z0-061.10.4.2)

Which `CREATE TABLE` statements will fail? (Choose all that apply.)

- ☐ `CREATE TABLE time1 (time1 NUMBER(9));`
- ☐ `CREATE TABLE date (time_id NUMBER(9));`
- ☐ `CREATE TABLE time (time_id NUMBER(9));`
- ☐ `CREATE TABLE time* (time_id NUMBER(9));`
- ☐ `CREATE TABLE $time (time_id NUMBER(9));`
- ☐ `CREATE TABLE datetime (time_id NUMBER(9));`

Answer:

`CREATE TABLE date (time_id NUMBER(9));`
`CREATE TABLE time* (time_id NUMBER(9));`
`CREATE TABLE $time (time_id NUMBER(9));`

Explanation:

The following `CREATE TABLE` statements will fail:

```
CREATE TABLE date (time_id NUMBER(9));
CREATE TABLE time* (time_id NUMBER(9));
CREATE TABLE $time (time_id NUMBER(9));
```

The `CREATE TABLE` statement for the `date` table will return an `ORA-00903: invalid table name` error because `DATE` is an Oracle Server reserved word. The `CREATE TABLE` statement for the `time*` table will return an `ORA-00922: missing or invalid option` error because database tables and column names cannot contain the `*` character. The `CREATE TABLE` statement for the `$time` table will return an `ORA-00911: invalid character` error because table and column names must begin with a letter.

The `CREATE TABLE` statement for the `time1` table will succeed. Table names must not duplicate the name of another object owned by the same Oracle Server user, but this rule does not apply to column names.

The `CREATE TABLE` statements for the `time` and `datetime` tables will succeed because neither `time` nor `datetime` are considered to be Oracle Server reserved words, and each table name follows the standard database object-naming conventions.

Item: 3 (Ref:1Z0-061.10.4.3)

Evaluate this `CREATE TABLE` statement.

```
SQL> CREATE TABLE order*1 (
order# NUMBER(9),
cust_id NUMBER(9),
date_1 DATE DEFAULT SYSDATE);
```

Which line of this statement will cause an error?

- ○ Line 1
- ○ Line 2
- ○ Line 3
- ○ Line 4

Answer:

Line 1

Explanation:

The first line of this `CREATE TABLE` statement will cause an `ORA-00903: invalid table name` error because `order*1` is not a legal table name. Table and column names cannot contain the `*` character. Table and column names can only contain the characters A-Z, a-z, 0-9, _ (underscore), $, and #. Oracle discourages the use of the $ and # characters.

The column names `order#`, `cust_id`, and `date_1` are all legal column names. Each begins with a letter, is 1-30 characters long, contains only legal characters, and is not an Oracle Server reserved word.

Item: 4 (Ref:1Z0-061.10.5.3)

Examine the structure of the `employee` table.

`EMPLOYEE` table (structure)

COLUMN_NAME	DATATYPE	CONSTRAINT
`EMPLOYEE_ID`	`NUMBER`	NOT NULL, Primary Key
`EMP_LNAME`	`VARCHAR2(25)`	
`EMP_FNAME`	`VARCHAR2(25)`	
`DEPT_ID`	`NUMBER`	Foreign key to `DEPT_ID` column of `DEPARTMENT` table
`JOB_ID`	`NUMBER`	Foreign key to `JOB_ID` column of `JOB` table
`MGR_ID`	`NUMBER`	Foreign Key to `EMPLOYEE_ID` column of `EMPLOYEE` table (this one)
`SALARY`	`NUMBER(9,2)`	
`HIRE_DATE`	`DATE`	

Which `CREATE TABLE` statement should you use to create the `employee` table?

○
```
CREATE TABLE employee (
employee_id NUMBER,
emp_lname VARCHAR2(25) NOT NULL,
emp_fname VARCHAR2(25),
dept_id NUMBER,
job_id NUMBER,
mgr_id NUMBER,
salary NUMBER(9,2),
hire_date DATE,
CONSTRAINT employee_id_pk PRIMARY KEY(employee_id));
```

○
```
CREATE TABLE employee (
employee_id NUMBER,
emp_lname VARCHAR2(25) NOT NULL,
emp_fname VARCHAR2(25),
dept_id NUMBER,
job_id NUMBER,
mgr_id NUMBER,
salary NUMBER(9,2),
hire_date DATE,
CONSTRAINT employee_id_pk PRIMARY KEY(employee_id),
CONSTRAINT mgr_id_fk FOREIGN KEY(mgr_id) REFERENCES employee(employee_id));
```

○
```
CREATE TABLE employee (
employee_id NUMBER,
emp_lname VARCHAR2(25) NOT NULL,
emp_fname VARCHAR2(25),
dept_id NUMBER,
job_id NUMBER,
mgr_id NUMBER,
salary NUMBER(9,2),
hire_date DATE,
CONSTRAINT employee_id_pk PRIMARY KEY(employee_id),
CONSTRAINT dept_id_fk FOREIGN KEY(dept_id) REFERENCES department(dept_id),
CONSTRAINT job_id_fk FOREIGN KEY(job_id) REFERENCES job(job_id));
```

○
```
CREATE TABLE employee (
employee_id NUMBER,
emp_lname VARCHAR2(25) NOT NULL,
emp_fname VARCHAR2(25),
dept_id NUMBER,
job_id NUMBER,
mgr_id NUMBER,
salary NUMBER(9,2),
hire_date DATE,
CONSTRAINT employee_id_pk PRIMARY KEY(employee_id),
CONSTRAINT dept_id_fk FOREIGN KEY(dept_id) REFERENCES department(dept_id),
CONSTRAINT job_id_fk FOREIGN KEY(job_id) REFERENCES job(job_id),
CONSTRAINT mgr_id_fk FOREIGN KEY(mgr_id) REFERENCES employee(employee_id));
```

Answer:

```
CREATE TABLE employee (
employee_id NUMBER,
emp_lname VARCHAR2(25) NOT NULL,
emp_fname VARCHAR2(25),
dept_id NUMBER,
job_id NUMBER,
mgr_id NUMBER,
salary NUMBER(9,2),
hire_date DATE,
CONSTRAINT employee_id_pk PRIMARY KEY(employee_id),
CONSTRAINT dept_id_fk FOREIGN KEY(dept_id) REFERENCES department(dept_id),
CONSTRAINT job_id_fk FOREIGN KEY(job_id) REFERENCES job(job_id),
CONSTRAINT mgr_id_fk FOREIGN KEY(mgr_id) REFERENCES employee(employee_id));
```

Explanation:
You should use the following `CREATE TABLE` statement to create the `employee` table:

```
CREATE TABLE employee (
employee_id NUMBER,
emp_lname VARCHAR2(25) NOT NULL,
emp_fname VARCHAR2(25),
dept_id NUMBER,
job_id NUMBER,
mgr_id NUMBER,
salary NUMBER(9,2),
hire_date DATE,
CONSTRAINT employee_id_pk PRIMARY KEY(employee_id),
CONSTRAINT dept_id_fk FOREIGN KEY(dept_id) REFERENCES department(dept_id),
CONSTRAINT job_id_fk FOREIGN KEY(job_id) REFERENCES job(job_id),
CONSTRAINT mgr_id_fk FOREIGN KEY(mgr_id) REFERENCES employee(employee_id));
```

The statement required to create the `employee` table contains five `CONSTRAINT` clauses. A primary key constraint, a `NOT NULL` constraint, and three foreign key constraints are required. The clause `CONSTRAINT employee_id_pk PRIMARY KEY(employee_id)` creates a primary key constraint on the `employee_id` column. This primary key constraint enforces uniqueness of the values in the `employee_id` column.

You do not need to specify a `NOT NULL` constraint on this column because a primary key constraint ensures that no part of the primary key can contain a null value.

The clause `NOT NULL` on the `emp_lname` column will create an Oracle named `NOT NULL` constraint on that column.

The clause `CONSTRAINT dept_id_fk FOREIGN KEY(dept_id) REFERENCES department(dept_id)` creates a foreign key constraint on the `dept_id` column in the `employee` table that references the `dept_id` column in the `department` table.

The clause `CONSTRAINT job_id_fk FOREIGN KEY(job_id) REFERENCES job(job_id)` creates a foreign key constraint on the `job_id` column in the `employee` table that references the `job_id` column in the `department` table.

The clause `CONSTRAINT mgr_id_fk FOREIGN KEY(mgr_id) REFERENCES employee(employee_id)` creates a foreign key constraint on the `mgr_id` column in the `employee` table that references the `employee_id` column in the `employee` table. These primary key constraints ensure that all values in the `dept_id`, `job_id`, and `mgr_id` columns in the `employee` table match an existing value in the parent table or have a null value.

All of the other options are incorrect because they do not satisfy the `employee` table requirements.

Item: 5 (Ref:1Z0-061.10.4.1)

Evaluate this `CREATE TABLE` statement:

```
CREATE TABLE customer (
customer_id NUMBER,
company_id VARCHAR2(30),
contact_name VARCHAR2(30),
contact_title VARCHAR2(20),
address VARCHAR2(30),
city VARCHAR2(25),
region VARCHAR2(10),
postal_code VARCHAR2(20),
country_id NUMBER DEFAULT 25,
phone VARCHAR2(20),
fax VARCHAR2(20),
credit_limit NUMBER (7,2));
```

Which three business requirements will this statement accomplish? (Choose three.)

- ☐ Credit limit values can be up to $1,000,000.
- ☐ Company identification values could be either numbers or characters, or a combination of both.
- ☐ All company identification values are only 6 digits so the column should be fixed in length.
- ☐ Phone number values can range from 0 to 20 characters so the column should be variable in length.
- ☐ The value 25 should be used if no value is provided for the country identification when a record is inserted.

Answer:

Company identification values could be either numbers or characters, or a combination of both.
Phone number values can range from 0 to 20 characters so the column should be variable in length.
The value 25 should be used if no value is provided for the country identification when a record is inserted.

Explanation:

The given `CREATE TABLE` statement will accomplish the following three business requirements:

- Company identification values could be either numbers or characters, or a combination of both.
- Phone number values can range from 0 to 20 characters so the column should be variable in length.
- The value 25 should be used if no value is provided for the country identification when a record is inserted.

Defining the `company_id` column as a `VARCHAR2` data type allows for character data, including A-Z, a-z, and 0-9. Defining the `PHONE` column in the `customer` table as a `VARCHAR2` data type provides for variable-length character data. In this example, phone number values can be up to 20 characters in length, but the size of the column will vary depending on each row value. Defining the `COUNTRY_ID` column with the `DEFAULT` option with a value of 25 ensures that the value of 25 will be used in `INSERT` operations if no value is provided. The `DEFAULT` option will prevent a null value from being inserted into the `COUNTRY_ID` column if a row is inserted without a `COUNTRY_ID` value.

Defining the `CREDIT_LIMIT` column of data type `NUMBER`(7,2) allows values up to 99,999.99. The column is defined with a precision of 7 and a scale of 2. If the user attempts to insert a credit limit value with more than 5 digits to the left of the decimal, an `ORA-01438: value larger than specified precision allows for this column` error would be returned.

Defining the `company_id` value of data type `VARCHAR2`(30) creates a variable-length character column of 30 bytes. Therefore, the business rule requiring a six-byte fixed-length column is not met.

Item: 6 (Ref:1Z0-061.10.3.3)

You need to create the `ELEMENT` table. The atomic weights of elements have varying decimal places. For example, values could be 4, 4.35, or 4.3567.

Which data type would be most appropriate for the atomic weight values?

- ○ `RAW`
- ○ `LONG`
- ○ `NUMBER`
- ○ `NUMBER(p,s)`

Answer:

`NUMBER`

Explanation:

The `NUMBER` data type without precision or scale would be most appropriate. When a fixed-point number is not known, as in this example, floating-point numbers should be used. Floating-point numbers are represented using the `NUMBER` data type.

All of the other options are incorrect. The `NUMBER` data type with precision and scale (`p,s`) stores fixed-point numbers. The `LONG` data type stores character data of variable length up to 2 gigabytes. The `RAW` data type stores raw binary data of a specified size.

Item: 7 (Ref:1Z0-061.10.6.1)

You need to add a foreign key constraint to the `LINE_ITEM_ID` column in the `CURR_ORDER` table to refer to the `ID` column in the `LINE_ITEM` table.

Which statement should you use to achieve this result?

○
```
ALTER TABLE curr_order
ADD CONSTRAINT line_itemid_fk FOREIGN KEY (line_item_id) REFERENCES line_item (id);
```
○
```
ALTER TABLE curr_order
MODIFY (CONSTRAINT lineitem_id_fk FOREIGN KEY (line_item_id)
REFERENCES line_item (id));
```
○
```
ALTER TABLE line_item
ADD CONSTRAINT curr_order_line_item_id_fk FOREIGN KEY (id)
REFERENCES curr_order (line_item_id);
```
○
```
ALTER TABLE line_item
MODIFY (CONSTRAINT curr_order_lineitem_fk FOREIGN KEY (id)
REFERENCES curr_order (line_item_id));
```

Answer:
```
ALTER TABLE curr_order
ADD CONSTRAINT line_itemid_fk FOREIGN KEY (line_item_id) REFERENCES line_item (id);
```

Explanation:

You should use the following statement:

```
ALTER TABLE curr_order
ADD CONSTRAINT line_itemid_fk FOREIGN KEY (line_item_id) REFERENCES line_item (id);
```

To add a foreign key constraint to an existing column in a table, use the `ALTER TABLE` statement with the `ADD` clause. In this scenario, the `ALTER TABLE` command adds a foreign key constraint to the `LINE_ITEM_ID` column in the `CURR_ORDER` table. This foreign key constraint references the `ID` column in the `LINE_ITEM` table.

The `ALTER TABLE line_item` statement is incorrect because it will add a foreign key on the `ID` column in the `LINE_ITEM` table. The desired result is to add the constraint to the `CURR_ORDER` table.

The `ALTER TABLE` statements using the `MODIFY` clause are incorrect because you cannot add a foreign key constraint using this clause. These statements will return a syntax error.

Item: 8 (Ref:1Z0-061.10.3.1)

Which two statements about a column are true? (Choose two.)

- ☐ You cannot decrease the width of a `CHAR` column.
- ☐ You cannot increase the width of a `VARCHAR2` column.
- ☐ You cannot convert a `CHAR` data type column to the `VARCHAR2` data type.
- ☐ You cannot specify the column's position when adding a new column to a table.
- ☐ You cannot modify the data type of a column if the column contains non-null data.

Answer:

You cannot specify the column's position when adding a new column to a table.
You cannot modify the data type of a column if the column contains non-null data.

Explanation:
The following two statements about a column are true:

- You cannot specify the column's position when adding a new column to a table.
- You cannot modify the data type of a column if the column contains non-null data.

You can add or modify a column using the `ALTER TABLE` statement. When you add a column, you cannot specify the column's position. The new column automatically becomes the last column. You can modify a column's data type, size, or default value, but you can only change a column's data type if the column contains only null values or if the table is empty.

All of the other options are incorrect. You can decrease the width of a column if the column contains only null values or if the table is empty. You can increase the width of a column. You can convert a `CHAR` data type column to the `VARCHAR2` data type if the column contains null values or if you do not change the column size.

Item: 9 (Ref:1Z0-061.10.5.4)

You disabled the primary key constraint on the `ID` column in the `INVENTORY` table and updated all the values in the `INVENTORY` table. You need to enable the constraint and verify that the new `ID` column values do not violate the constraint. If any of the `ID` column values do not conform to the constraint, an error message should be returned.

Evaluate this statement:

```
ALTER TABLE inventory
ENABLE CONSTRAINT inventory_id_pk;
```

Which statement is true?

- ◯ The statement will achieve the desired results.
- ◯ The statement will execute, but will not enable the `PRIMARY KEY` constraint.
- ◯ The statement will execute, but will not verify that values in the ID column do not violate the constraint.
- ◯ The statement will return a syntax error.

Answer:

The statement will achieve the desired results.

Explanation:

The statement will achieve the desired results. In this scenario, the `ALTER TABLE ENABLE CONSTRAINT` statement will activate the currently disabled primary key constraint on the `INVENTORY` table. When a primary key constraint is disabled, the corresponding unique indexes for the constraint are automatically dropped. When the constraint is enabled, the constraint will apply to all the data in the table.

If existing data in the table does not adhere to the constraint, the `ALTER TABLE ENABLE CONSTRAINT` statement will fail and generate an error message. When you enable the primary key constraint, a primary key index is automatically created.

The remaining options are incorrect because the statement will execute and the desired results are achieved.

Item: 10 (Ref:1Z0-061.10.3.2)

Which statements about data types are true? (Choose all that apply.)

- ☐ The `BLOB` data type stores character data up to four gigabytes.
- ☐ The `TIMESTAMP` data type is an extension of the `VARCHAR2` data type.
- ☐ The `CHAR` data type should be used for fixed-length character data.
- ☐ The `INTERVAL YEAR TO MONTH` data type allows time to be stored as an interval of years and months.
- ☐ The `VARCHAR2` data type requires that a minimum size be specified when defining a column of this type.

Answer:

The `CHAR` data type should be used for fixed-length character data.

The `INTERVAL YEAR TO MONTH` data type allows time to be stored as an interval of years and months.

Explanation:

The `CHAR` data type should be used for fixed-length character data. The length of the column is specified in bytes. The minimum size is 1, and the maximum size is 2000.

The `INTERVAL YEAR TO MONTH` data type stores a period of time using the `YEAR` and `MONTH` datetime fields. For example, `INTERVAL '315-5' YEAR(3) TO MONTH` indicates an interval of 315 years and 5 months.

The `TIMESTAMP` data type is an extension of the `DATE` data type and is used to store time as a date with fractional seconds.

The `BLOB` data type stores binary data up to four gigabytes, and the CLOB data type stores character data up to four gigabytes.

The `VARCHAR2` data type does not require that a minimum size be specified, but it does require that a maximum size be specified.

Item: 11 (Ref:1Z0-061.10.5.2)

Which `ALTER TABLE` statement should you use to add a `PRIMARY KEY` constraint on the `manufacturer_id` column of the `INVENTORY` table?

- ○
```
ALTER TABLE inventory
ADD PRIMARY KEY (manufacturer_id);
```
- ○
```
ALTER TABLE inventory
ADD CONSTRAINT manufacturer_id PRIMARY KEY;
```
- ○
```
ALTER TABLE inventory
MODIFY manufacturer_id CONSTRAINT PRIMARY KEY;
```
- ○
```
ALTER TABLE inventory
MODIFY CONSTRAINT PRIMARY KEY manufacturer_id;
```

Answer:

```
ALTER TABLE inventory
ADD PRIMARY KEY (manufacturer_id);
```

Explanation:

You should use the following `ALTER TABLE` statement to add a primary key constraint on the `manufacturer_id` column of the `INVENTORY` table:

```
ALTER TABLE inventory
ADD PRIMARY KEY (manufacturer_id);
```

To add a primary key constraint to an existing column in a table, use the `ALTER TABLE` statement with the `ADD` clause. In this scenario, the `ALTER TABLE` statement adds a primary key constraint to the `manufacturer_id` column in the `INVENTORY` table. Because a name is not provided for the primary key constraint, a system-generated name will be used. To specify a constraint name, you must use the `CONSTRAINT` keyword in the `ADD` clause. For this statement to execute, the table must be empty or all the values in the `manufacturer_id` column must be unique and non-null.

The `ALTER TABLE` statements using the `MODIFY` clause are incorrect because you cannot add a primary key constraint using this clause. These statements will return a syntax error.

The `ALTER TABLE` statement containing the clause `ADD CONSTRAINT manufacturer_id PRIMARY KEY` will return a syntax error because the column name for the primary key constraint is not specified.

Item: 12 (Ref:1Z0-061.10.5.1)

Study the `PHYSICIAN` table before answering the following question.

PHYSICIAN table (structure)

COLUMN_NAME	DATATYPE	CONSTRAINT
CATEGORY_ID	NUMBER(9)	Primary Key
DESCRIPTION	VARCHAR2(50)	NOT NULL
COST	NUMBER(9,2)	NOT NULL
PRICE	NUMBER(9,2)	

PHYSICIAN table (data)

PHYSICIAN_ID	FIRST_NAME	LAST_NAME	LICENSE_NO
2135	John	Smith	7500021
4773	Marsha	Lane	2665030
9055	Fredrick	Jones	
1777	Sasha	Patel	7500021

Physicians are assigned a unique physician ID and a license number based upon their specialty. For example, if two doctors are both pediatricians, they will have different physician IDs but the same license number. Not all physicians have a license number.

You want to create a `SELECT` statement that will prompt the user for a specific license number. You should return the IDs and names of all physicians who do NOT have that specific license number, including names and IDs of physicians with no license number.

Which of the following `SELECT` statements will meet the scenario requirements? (Choose all that apply.)

- ☐ `SELECT physician_id, first_name||' '||last_name "Name" FROM physician WHERE license_no NOT IN (&license_number)`
- ☐ `SELECT physician_id, first_name||' '||last_name "Name" FROM physician MINUS SELECT physician_id, first_name||' '||last_name FROM physician WHERE license_no = &license_number`
- ☐ `SELECT physician_id, first_name||' '||last_name "Name" FROM physician WHERE license_no <> &license_number`
- ☐ `SELECT physician_id, first_name||' '||last_name "Name" FROM physician WHERE license_no != &license_number or license_no IS NULL`

Answer:

`SELECT physician_id, first_name||' '||last_name "Name" FROM physician MINUS SELECT physician_id, first_name||' '||last_name FROM physician WHERE license_no = &license_number`

`SELECT physician_id, first_name||' '||last_name "Name" FROM physician WHERE license_no != &license_number or license_no IS NULL`

Explanation:

The two statements that meet the scenario requirements are:

```
SELECT physician_id, first_name||' '||last_name "Name" FROM physician MINUS SELECT physician_id, first_name||'
'||last_name FROM physician WHERE license_no = &license_number

SELECT physician_id, first_name||' '||last_name "Name" FROM physician WHERE license_no != &license_number or
license_no IS NULL
```

The `SELECT` statement that contains the `MINUS` operator is one of the correct answers. The `SELECT` prior to the keyword `MINUS` returns all the rows in the table, regardless of whether the column `license_no` is `NULL` or not, and regardless of the value the user entered when prompted. Then, the `SELECT` after the keyword `MINUS` removes the rows where the `license_no` is equal to the value entered by the user. Consequently, you will end up with all rows except those with the value entered by the user, as well as rows in which the `license_no` is null.

The `SELECT` statement that contains the operator `!=` is also correct. It will return all the rows in which the `license_no` is not equal to the value entered by the user, as well as rows in which the value of `license_no` is null.

The `SELECT` statement that contains the `NOT IN` operator is incorrect. When the user enters the value for `license_no`, the software determines where there is a mismatch between the value the user entered and the value for `license_no`. When it gets to the row(s) where `license_no` is null, it will be unable to evaluate whether the `WHERE` clause is true or false. Since the value of the condition is unknown (`NULL`) and NOT true, the row with the null value will never be returned. Consequently, this `SELECT` statement will miss returning the rows where the value of `license_no` is null.

The `SELECT` statement that contains the <> operator is also incorrect. The software will not evaluate the condition as being TRUE when it encounters a row where the value of the column is null. Hence, those rows will not be returned, and thus the requirements of the scenario are not met.

1Z0-061: DDL

Item: 13 (Ref:1Z0-061.10.3.4)

Which data type is a hexadecimal string representing the unique address of a row in its table?

- ○ `RAW`
- ○ `ROWID`
- ○ `BFILE`
- ○ `VARCHAR2`

Answer:

`ROWID`

Explanation:

The `ROWID` data type is a hexadecimal string that represents the unique address of a row in its table. `ROWID` values can be queried just like other values in a table. Because `ROWID` provides a unique identifier for each row in the table, it can be used to locate a row in a table.

The `RAW` data type allows for the storage of binary data of a specified size. When data is stored as type `RAW` or `LONG RAW`, the Oracle Server does not perform character set conversion when the data is transmitted across machines in a network or when data is moved from one database to another.

The `BFILE` data type stores unstructured data in operating system files.

The `VARCHAR2` data type stores variable-length character data and uses only the number of bytes needed to store the actual column value.

Item: 14 (Ref:1Z0-061.10.2.1)

Which is a legal table name?

- ○ `COLUMN`
- ○ `NUMBER#1`
- ○ `1CUSTOMER`
- ○ `NEW_CUSTOMER_ENTERED_BEFORE_SEPT01`

Answer:

`NUMBER#1`

Explanation:

`NUMBER#1` is a legal table name because it begins with a letter, is 1-30 characters long, and contains only characters A-Z, a-z, 0-9, _ (underscore), $, and #. Although `NUMBER` is an Oracle Server reserved word, `NUMBER#1` is not considered to be a reserved word.

All of the other options are incorrect. `COLUMN` is not a legal table name because it is an Oracle Server reserved word. `1CUSTOMER` is not a legal table name because it does not begin with a letter. Attempting to create a table named `COLUMN` or `1CUSTOMER` will return an `ORA-00903: invalid table name` error.

`NEW_CUSTOMER_ENTERED_BEFORE_SEPT01` is not a legal table name because it is longer than 30 characters. Attempting to create a table with a table name longer than 30 characters will return an `ORA-00972: identifier is too long` error.