Question 1

In which of the following scenarios a data engineer would use an all-purpose cluster?

- **When the cluster needs to be shared between multiple users**

  (Correct)

- **When the data engineer needs to save the cost**

- **When the data engineer needs to schedule the job to run every hour**

- **When the job contains multiple languages**

- **When the data engineer needs to terminate the cluster as the job ends**

Explanation

There are two types of **clusters** in Databricks. A **job cluster** and an **all-purpose cluster**. An *all-purpose cluster* can be shared between users while a *job cluster* cannot be shared between users.

An *all-purpose cluster* is created by the **user** specifying the required settings but the *job cluster* is created as soon as you run a job. The job cluster is **terminated** as soon as the **job finishes**.

More Info: Choosing the correct type of cluster in Databricks

Question 2

Which of the following define the managed table correctly?

- **Managed tables are those which are created by the Databricks admin**
- **Managed tables are those for which both metadata and data are managed by Databricks**

  (Correct)
- **Managed tables are those for which only metadata is managed by Databricks**
- **Managed tables are those which are dropped automatically after use**
- **Managed tables are those for which only data is managed by Databricks**

Explanation

Let us explore all the options and try to find out which one could be true for a managed table.

Managed tables are those which are created by the Databricks admin

INCORRECT. The **type of table** does not depend on the **user** who created the table

Managed tables are those for which both metadata and data are managed by Databricks

CORRECT. As the name suggests, a table for which both **metadata and data are managed by Databricks** is called a **managed** table.

Managed tables are those for which only metadata is managed by Databricks

INCORRECT. As discussed earlier, **Databricks manages both data and metadata** for a **managed** table.

Managed tables are those which are dropped automatically after use

INCORRECT. There is **no table type** in Databricks which are **dropped automatically** after use.

Managed tables are those for which only data is managed by Databricks

INCORRECT. As discussed earlier, **Databricks manages both data and metadata** for a **managed** table.

Also note, for external or unmanaged tables Databricks manages only the metadata associated with the table.

More Info: Managed table in Databricks

Question 3

Which of the following commands can be used to combine small files to a bigger file to achieve better performance?

- `VACUUM`
- `DELETE`
- `OPTIMIZE`

  (Correct)
- `COMBINE`
- `RESTORE`

Explanation

Some of the above commands can be asked in the actual exam. So, it is preferred that you know each one of them. Let us go through all the options and learn more about their usage to help you train for the actual exam.

`VACUUM` – **Cleans** up files that are older than X **number of hours**.

`DELETE` – Used to delete data from a table

`OPTIMIZE` – Used for **optimizing** the files by combining **small files to a bigger file**.

`COMBINE` – Invalid command

`RESTORE` – Used for restoring a table to a **previous version** or **timestamp**

Looking at all the options and their usage you now know why `OPTIMIZE` is the correct answer.

More Info: VACUUM | DELETE FROM | OPTIMIZE | RESTORE

Question 4

In a notebook, all the cells contain code in Python language and you want to add another cell with a SQL statement in it. You changed the default language of the notebook to accomplish this task. What changes (if any) can be seen in the already existing Python cells?

- **The Python cells will be grayed out and won't run until you change the default language back to Python**
- **The magic command `%python` will be added at the beginning of all the cells that contain Python code**

  (Correct)
- **The magic command `%python` will be added at the end of all the cells that contain Python code**
- **The magic command `%sql` will be added at the beginning of all the cells that contain Python code**
- **There will be no change in any cell and the notebook will remain the same**

Explanation

In this case, the magic command `%python` will be added at the **beginning of all the cells** that contain Python code. If you try to change the default language for a notebook, all the cells containing code in previously selected default language will be updated and `%previous_default_language` (like `%sql`, `%scala`) will be added as the **first line** in the cells. This is done to assure that the **default language changes** will **not affect** the working of the notebook.

The following are the **language magic commands** supported in Databricks notebook:

1. `%scala`
2. `%r`
3. `%python`
4. `%sql`

More Info: Effect of changing the default language of a Databricks notebook

Question 5

Which of the following statements is INCORRECT for a Delta Lake?

- **Delta Lake is open-source**
- **Delta Lake is ACID compliance**
- **Delta Lake can run on an existing Data Lake**
- **Delta Lake is compatible with Spark API**
- **Delta Lake stores data with `.dl` extension**

  (Correct)

Explanation
Let us go through all the options one by one and find out which one isn't correct for a Delta Lake.

Delta Lake is open-source - Delta Lake is an **open-source storage framework** which frames the base for the Databricks **Lakehouse** platform.

Delta Lake is ACID compliance - Delta Lake ensures **consistent** data for the readers while making sure that multiple reads and writes to the table are **conflict-less**.

Delta Lake can run on an existing Data Lake - Delta Lake runs on top of your **existing Data Lake**.

Delta Lake is compatible with Spark API - Delta Lake is **fully** compatible with **Spark API.**

Delta Lake stores data with `.dl` extension - Delta Lake does **not** store data with `.dl` extension but stores it in **Parquet format with versioning**. Delta Lake also stores **transaction logs** known as `DeltaLog`

It is important to note that Delta Lake is the default for all tables created in Databricks which means adding `USING DELTA` while creating a table is redundant.

More Info: Delta Lake

Question 6

A data engineer needs to create a Delta table `employee` only if a table with the same name does not exist. Which of the following SQL statements can be used to create the table?

- ○
```
1. CREATE OR REPLACE TABLE employee
2. (emp_id int, name string)
3. USING DELTA;
```
- ○
```
1. CREATE TABLE employee
2. (emp_id int, name string);
```
- ○
```
1. CREATE TABLE employee IF NOT EXISTS
2. (emp_id int, name string)
3. USING DELTA;
```
- ◉
```
1. CREATE TABLE IF NOT EXISTS employee
2. (emp_id int, name string);
```

(Correct)

- ○
```
1. CREATE OR REPLACE TABLE employee
2. emp_id int, name string;
```

Explanation

`CREATE TABLE IF NOT EXISTS` is used when you need to create a table **ONLY** if it does not exist. So, all the options that do not use `CREATE TABLE IF NOT EXISTS` are incorrect. We are down to just 2 options, let's examine each of them.

```
1. CREATE TABLE employee IF NOT EXISTS
2. (emp_id int, name string)
3. USING DELTA;
```

INCORRECT. As per the syntax of table creation, `IF NOT EXISTS` should come **before** the table name.

```
1. CREATE TABLE IF NOT EXISTS employee
2. (emp_id int, name string);
```

CORRECT. This is the correct code block. It will create the table **only if it does not exist**. Moreover, the default table type in Databricks is `DELTA` which means that adding `USING DELTA` at the end of the query is optional.

More Info: Using IF NOT EXISTS for table creation

Question 7

Which of the following is the correct order of git commands to save the data in the central repository?

- `git add -> git pull`
- `git add -> git push -> git commit`
- `git push -> git add -> git commit`
- `git add -> git commit -> git push`

  (Correct)
- `git pull -> git commit -> git push`

Explanation
The correct order of commands is as follows:

```
git add -> git commit -> git push
```

`git add` adds the files to a **staging area** while `git commit` is used for **recording the changes** in the repository. `git push` is used to push the changes with its logs to a **central repository**.

Also note, if you need to get a **remote repository** in Databricks, you would need to **clone** the repository first. To get the **latest updates** from a remote repository, you can use `git pull` which makes your local repo in **sync** with the remote repo.

More Info: Basic git commands

Question 8

A junior data engineer has just joined your team and has accidentally deleted all the records from delta table `currency_exchange` using the command `DELETE FROM currency_exchange`.

You ran the following query: `DESCRIBE HISTORY currency_exchange` and found out that the latest version is 6. You need to roll back the delete operation and get the contents back. Which of the following SQL statements will accomplish this task?

- ```
  RESTORE TABLE currency_exchange TO VERSION AS OF 6
  ```
- ```
  RESTORE TABLE currency_exchange TO VERSION 5
  ```
- ```
  ROLLBACK currency_exchange TO VERSION AS OF 5
  ```
- ```
  RESTORE TABLE currency_exchange TO VERSION AS OF 5
  ```

  (Correct)
- ```
  RESTORE currency_exchange TO VERSION 6
  ```

Explanation
The
  ```

Question states that the latest version is 6 which means that you need to restore the table to version 5 which still has all the deleted data. Now, the syntax to restore a table is `RESTORE TABLE [table_name] TO VERSION AS OF [version]`

As you need to restore it to version 5 your query will look like this:

```
RESTORE TABLE currency_exchange TO VERSION AS OF 5
```

More Info: Restoring a table to an earlier state

Question 9

As a data engineer, you dropped a table by using `DROP` command and noticed that the table has been removed but the underlying data is still present. Which of the following is true about the situation?

- **The DROP command has not been completed successfully**
- **The table is a managed table**
- **The data of the table is delete-protected**
- **The table is an external or unmanaged table**

  (Correct)
- **The table is a semi-managed table**

Explanation

Firstly, you should know the difference between a **managed** and an **unmanaged** table.

A table for which **both data and metadata** are controlled by Databricks is known as a **managed table**. When you **drop** a managed table, **data** is also **deleted**.

The other type of table is an **unmanaged or external table** which is being referred to in this

Question. The unmanaged table gives the control of the data to the user and **only the metadata is controlled by Databricks**. Considering these facts, you can easily judge the right answer.

More Info: Managed and unmanaged tables in Databricks

Question 10

Which of the following pages will allow you to view, update or delete a cluster?

- **Data Page**
- **Compute Page**

  (Correct)
- **Workflows Page**
- **Experiments Page**
- **Functions Page**

Explanation

The left side navigation menu is the **most often** used thing in the Databricks. It can be used to switch between **Data Science and Engineering**, **Machine Learning** and **Databricks SQL** as per the requirement.

The **Compute page or pane** allows users to examine **cluster** related operations. Some of the tasks that can be done on the **Compute page or pane** are as follows:

1. Create a new cluster

2. Delete an existing cluster

3. Stop a running cluster

4. Start a cluster

5. View the list of clusters

and more...

Now, let us take a look at the other options given in the

Question.

Data page or pane - It contains **DBFS** and **Database tables** tabs. It is mainly used for accessing the **files** stored on DBFS or to quickly look at the **databases** and **tables**.

Workflows page or pane - It is used for creating **jobs, tasks, workflows, ETL pipelines** etc.

Function page or pane - There is **no** such page or pane in Databricks.

Experiments page or pane - It is used to create, view, update and delete **Machine Learning Experiments.**

More Info: Creating clusters in Compute page

Question 11

Databricks allows users to change the default language of their notebooks. Which of the following languages cannot be set as a default language?

- **Scala**
- **Java**

  (Correct)
- **Python**
- **SQL**
- **R**

Explanation

Databricks supports **Scala**, **Python**, **SQL** and **R** as default languages while **Java is not supported** as of now. To change the **default language** of a notebook, you need to select the required language from the **top of the notebook**. If you need to change the language of a particular **cell**, you can go to the top right corner of the cell and set it to your preferred language.

Remember, if you select a language for a cell which is not the default language of the notebook, the `%cell_language` will be added to the cell. So, if the default language of the notebook is Python and you select Scala for a particular cell, `%scala` will be added(as the first line) in the cell.

More Info: Default language in Databricks notebooks

Question 12

Which of the following commands can be used to rotate a table on one of its axes?

- `ROTATE`
- `PIVOT`

  (Correct)
- `FILTER`
- `EXIST`
- `REDUCE`

Explanation

`PIVOT` command helps in viewing different data perspectives of a table including **slicing, dicing and rotating**. Using `PIVOT` you can easily convert a **long table(more rows) to a wide table(more columns)**.

Let us understand this with an example.

The following table named `soccer` contains **3 columns** and **45 rows**.

	player	opposition	goals
1	Cristiano Ronaldo	Luxembourg	8
2	Cristiano Ronaldo	Sweden	7
3	Cristiano Ronaldo	Lithuania	7
4	Cristiano Ronaldo	Andorra	6
5	Cristiano Ronaldo	Latvia	5
6	Cristiano Ronaldo	Armenia	5
7	Cristiano Ronaldo	Switzerland	5
8	Cristiano Ronaldo	Hungary	4
9	Cristiano Ronaldo	Estonia	4
10	Cristiano Ronaldo	Netherlands	4
11	Cristiano Ronaldo	Faroe Islands	4
12	Cristiano Ronaldo	Spain	3
13	Cristiano Ronaldo	Russia	3
14	Cristiano Ronaldo	Belgium	3
15	Cristiano Ronaldo	Denmark	3
16	Cristiano Ronaldo	Northern Ireland	3
17	Cristiano Ronaldo	Egypt	2
18	Cristiano Ronaldo	Cyprus	2
19	Cristiano Ronaldo	France	2
20	Cristiano Ronaldo	Cameroooon	2
21	Cristiano Ronaldo	Azerbaijan	2
22	Cristiano Ronaldo	Kazakhstan	2
23	Cristiano Ronaldo	Saudi Arabia	2
24	Cristiano Ronaldo	Czech Republic	2
25	Cristiano Ronaldo	Republic of Ireland	2
26	Cristiano Ronaldo	Bosnia and Herzegovina	2
27	Cristiano Ronaldo	Iran	1
28	Cristiano Ronaldo	Qatar	1
29	Cristiano Ronaldo	Ghana	1
30	Cristiano Ronaldo	Wales	1
31	Cristiano Ronaldo	Serbia	1
32	Cristiano Ronaldo	Panama	1
33	Cristiano Ronaldo	Poland	1
34	Cristiano Ronaldo	Greece	1
35	Cristiano Ronaldo	Iceland	1
36	Cristiano Ronaldo	Ukraine	1
37	Cristiano Ronaldo	Morocco	1
38	Cristiano Ronaldo	Ecuador	1
39	Cristiano Ronaldo	Germany	1
40	Cristiano Ronaldo	Croatia	1
41	Cristiano Ronaldo	Finland	1
42	Cristiano Ronaldo	Slovakia	1
43	Cristiano Ronaldo	Argentina	1
44	Cristiano Ronaldo	New Zealand	1
45	Cristiano Ronaldo	North Korea	1

We, now run the following `PIVOT` query on the `soccer` table.

```
1. SELECT player, Germany, Argentina, Slovakia, Iceland, Greece, Wales, Iran,
   Spain, Netherlands, Sweden, Switzerland, Hungary
2. FROM soccer
3. PIVOT (SUM(goals) FOR opposition in ('Germany', 'Argentina', 'Slovakia',
   'Iceland', 'Greece', 'Wales', 'Iran', 'Spain', 'Netherlands', 'Sweden',
   'Switzerland', 'Hungary'))
```

This query results in the following table.

	player	Germany	Argentina	Slovakia	Iceland	Greece	Wales	Iran	Spain	Netherlands	Sweden	Switzerland	Hungary
1	Cristiano Ronaldo	1	1	1	1	1	1	1	3	4	7	5	4

You can notice that the **structure of the table has changed**. The values in the `opposition` column like Spain, Germany, Iceland are now placed as **individual columns**. This shows, you can easily convert a **long table(more rows) to a wide table(more columns)** using `PIVOT`.

Please note, this is just an example of how `PIVOT` method works. It does not justify the immense business applications of `PIVOT` operation.

More Info: PIVOT in SQL

Question 13

A data engineer is working on multiple Databricks notebooks. The next notebook to be run depends on the value of the python variable `next_notebook` which can range from 0 to 7. Which of the following can be used by the data engineer to run the notebooks depending on the value of `next_notebook`?

- **AutoLoader**
- **Multi-hop architecture**
- **SQL endpoint**
- **Python control flow**

  (Correct)
- **Delta lake**

Explanation

Python control flow uses `if else` statements to control the **flow of events**. It is similar to `CASE WHEN` statements in SQL. It can be used to decide which notebook should be running next based on the condition in the `if` statement. For instance, you want to run **setup notebook** if the value of `next_notebook` is 0, you need to give the following condition:

```
1. If next_notebook == 0:
2.     %run path_to_next_notebook
```

It allows you to **control** the **flow of events** based on the provided **conditions**.

More Info: Python Control Flow

Question 14

Which of the following statements will increase the `salary` by 10000 for all the employees that have `rating` greater than 3 in the `employees` table?

- ```
  1. UPDATE TABLE employees
  2. SET salary = salary + 10000
  3. WHERE rating > 3;
  ```
- ```
  1. UPDATE employees
  2. SET salary = salary + 10000
  3. WHERE rating > 3;
  ```

  (Correct)
- ```
  1. UPDATE employees
  2. SET salary = salary + 10000
  3. IF rating > 3;
  ```
- ```
  1. UPDATE employees TABLE
  2. SET salary = salary + 10000
  3. WHERE rating > 3;
  ```
- ```
  1. UPDATE employees
  2. salary = salary + 10000
  3. WHERE rating > 3;
  ```

Explanation

Syntax for `UPDATE` statement is:

```
1. UPDATE table_name
2. SET column = value….
3. WHERE condition;
```

Looking at the syntax, you can easily tell, the only option that satisfies the above syntax is option B, thus, is correct.

More Info: UPDATE command in SQL
```

Question 15

Find the error in the following SQL statement which intends to create a new database as per the following requirements. Also, if the database already exists an error message should be returned.

Database Name	Company
Description	Company Database
Location	/north/company

```
CREATE SCHEMA IF NOT EXISTS company COMMENT 'Company Database' LOCATION '/north/company'
```

- `CREATE SCHEMA` **should be replaced with** `CREATE DATABASE`
- `COMMENT` **is not a valid parameter,** `DESCRIPTION` **should be used**
- `IF NOT EXISTS` **should be removed from the statement**

  (Correct)
- **Name of the database should always be capitalized i.e.** `COMPANY`
- `CREATE` **should be appended with** `OR REPLACE`

Explanation

Let us scan all the options and try to find out which option is correct.

`CREATE SCHEMA` *should be replaced with* `CREATE DATABASE`

INCORRECT. `SCHEMA` and `DATABASE` can be used interchangeably.

`COMMENT` *is not a valid parameter,* `DESCRIPTION` *should be used*

INCORRECT. If you need to add some details about the newly created database, you can use `COMMENT` to do that and not `DESCRIPTION`

`IF NOT EXISTS` *should be removed from the statement*

CORRECT. As the

Question demands that an **error message** should be returned if the **database already exists**, `IF NOT EXISTS` should not be a part of the SQL statement as it will **not** return an error message when the database already exists.

Name of the database should always be capitalized i.e. `COMPANY`

INCORRECT. Database or table names are **not** case sensitive. Also note, even if you use `COMPANY` instead of `company` the query will still run successfully. Also, an interesting thing to note is that at **backend** all the databases and tables are stored in **small-case**. So, if you create a database named `CoUrSeS` it will still be stored as `courses` which you can confirm by running the following set of commands.

```
1  CREATE DATABASE CoUrSeS;
2  USE CoUrSeS;
3  SELECT current_database();
```

	current_database()
1	courses

`CREATE` *should be appended with* `OR REPLACE`

INCORRECT. If `CREATE OR REPLACE` is used in the query, the database will be **replaced** and **no error message** will be shown even if the database exists, which is definitely **not** required according to the

Question.

More Info: Using CREATE SCHEMA command to create new database

Question 16

The view `new_employees` contain a set of employees that joined the company in the past one week. Details for some of those employees have already been added to the `employees` table. Also, the view `new_employees` and table `employees` have the same schema. Which of the following SQL statements would insert only those records in the `employees` table which are not already present, checking on the basis of `emp_id` column?

- ```
  1. UPSERT INTO employees e
  2. USING new_employees n
  3. ON e.emp_id = n.emp_id
  4. WHEN MATCHED
  5. THEN INSERT *
  ```
- ```
  1. INSERT INTO employees VALUES (SELECT * FROM new_employees)
  ```
- ```
  1. MERGE INTO employees e
  2. USING new_employees n
  3. ON e.emp_id = n.emp_id
  4. WHEN MATCHED
  5. THEN INSERT ALL
  ```
- ```
  1. MERGE INTO employees e
  2. USING new_employees n
  3. ON e.emp_id = n.emp_id
  4. WHEN MATCHED
  5. THEN INSERT *
  ```

  (Correct)
- ```
  1. MERGE INTO employees e
  2. USING new_employees n
  3. ON e.emp_id = n.emp_id
  4. IF MATCHED
  5. THEN INSERT *
  ```

Explanation

In this type of
  ```

Questions, you need to use the `MERGE INTO` command. This command adds the rows based on the column given in the `ON` parameter. The correct code block is:

```
1. MERGE INTO employees e
2. USING new_employees n
3. ON e.emp_id = n.emp_id
4. WHEN MATCHED
5. THEN INSERT *
```

The `MERGE INTO` command is basically used for updating a table by **merging another table into it**. Usually, if the record does **not** already exist it is **added** and if the record is **already there** then it is **updated**. The `MERGE INTO` command can be used for **inserting**, **deleting** as well as **updating** the records.

More Info: Using MERGE INTO command for insertion, updation and deletion

Question 17

Which of the following views will be persisted through multiple sessions?

- **Only View**
- **View and Global Temporary View**

  (Correct)
- **View and Temporary View**
- **Global Temporary View**
- **All of them**

Explanation

There are **3 types** of Views supported in Databricks:

1. **View**

2. **Temporary View**

3. **Global Temporary View**

Out of these three - **View and Global Temporary View are persisted through multiple sessions** while the **Temporary View is tied to a session** and is not available to other sessions. Also note that if a cluster is restarted, the **temp views and the global temp views** are both **GONE** and **cannot** be accessed.

Remember that views do not have any physical existence. If you need to store a data object physically on the file system, you should consider creating a table instead.

More Info: Views in Databricks

Question 18

The following python function `show_str()` intends to print the string passed to the function. Find the error.

```
1. def show_str(string_to_show):
2.     print('string_to_show')
```

- **Python function is defined using `define` keyword and not `def`**
- **The colon should be removed**
- **`show()` function should be used instead of `print()`**
- **The quotes around the argument passed to `print()` function should be removed**

  (Correct)
- **The function has no errors, it will print the desired string**

Explanation

Let us look at all the options one by one to find out which part of the function(if any) needs to be changed.

Python function is defined using `define` keyword and not `def`

INCORRECT. In python programming language, function is defined using `def` keyword.

The colon should be removed

INCORRECT. Python does not use brackets to define the function body. Instead of brackets, it uses colon and indentation.

`show()` function should be used instead of `print()`

INCORRECT. If you need to print something on the screen in python, you would need to use `print()` function while `show()` is an operation on a PySpark DataFrame.

The quotes around the argument passed to `print()` function should be removed

CORRECT. If you try to call the above function, it will print – **string_to_show** instead of the **real string passed to the function**. So, you will need to **remove the quotes** to make python treat this as a **variable**.

The function has no errors, it will print the desired string

INCORRECT. As discussed above, the quotes should be removed.

More Info: Using print() function in python

Question 19

You have recently got to know about a directory that contains 108 parquet files. As a data engineer, you are asked to look at the first hundred rows from the data to check the quality of the data stored in the files. Which of the following SQL statements can be used?

- ```
  SELECT * FROM parquet.path LIMIT 100
  ```

- **You need to convert the files to a table as reading data directly from a directory is not supported in Databricks**

- ```
  SELECT * FROM parquet.`path` LIMIT 100
  ```

  (Correct)

- ```
  SELECT * FROM parquet.`path` FIRST 100
  ```

- ```
  SELECT * FROM path LIMIT 100
  ```

Explanation
This

Question tests your knowledge on **reading parquet files directly from a file system** like DBFS. Databricks supports viewing of **parquet files** in the form of a SQL table and allows the use of **filter** and other operations on a file or a directory.

Also, to answer this

Question you need to know which command should be used to find the **first N rows** from a SQL table. In SQL, `LIMIT` command is used followed by the number of rows you need to return in the result. So, the correct code block:

```
SELECT * FROM parquet.`path` LIMIT 100
```

Also note, **Backticks** are used around the **path** and **not the inverted commas**. In most keyboards, Backticks are found under the **Escape key** (with the **~** tilde symbol).

More Info: Reading from parquet files in Databricks

Question 20

The following is the employees table snapshot:

emp_code	details
126	{“name”: “Steve”, “age”, “23”, “salary”: “40000”}
165	{“name”: “Brad”, “age”, “27”, “salary”: “66400”}
. . .	. . .

The `details` column contains data in the form of JSON but the data type of the column is string. The data analyst wants to view the table in the following form:

emp_code	name	age	salary
126	Steve	23	40000
165	Brad	27	66400
...	...	...	...

Which of the following `SELECT` statements can be used to achieve the task?

- ```
  SELECT emp_code, EXPLODE(details)
  ```
- ```
  SELECT emp_code, details.*
  ```
- ```
  SELECT emp_code, details:*
  ```
- ```
  SELECT emp_code, details.name, details.age, details.salary
  ```
- ```
  SELECT emp_code, details:name, details:age, details:salary
  ```

  (Correct)

Explanation

Let us see all the options one by one to find the correct answer:

```
SELECT emp_code, EXPLODE(details)
```
```

INCORRECT. `EXPLODE` function can only be used on the **array or map type** columns but `details` is a **string** column. So, the `EXPLODE` function cannot be used on the `details` column.

```
SELECT emp_code, details.*
```

INCORRECT. `*` operator works with **struct** data type. Since, `details` is of string type `*` operator cannot be used.

```
SELECT emp_code, details:*
```

INCORRECT. As discussed earlier, `*` operator works with **struct** data type. Since, `details` is of **string** type `*` operator cannot be used.

```
SELECT emp_code, details.name, details.age, details.salary
```

INCORRECT. To unpack **JSON** value, `:` operator is used and not `.` operator.

```
SELECT emp_code, details:name, details:age, details:salary
```

CORRECT. This option ticks **all** the boxes to get the desired result.

More Info: Unpacking JSON column in SQL

Question 21

A data analyst needs to count the number of NULL values in column `secondary_mobile` from the data present in the `personal_details` table. Which of the following SQL statements, when executed, will fetch the required result?

- ```
  SELECT count(NULL secondary_mobile) FROM personal_details
  ```
- ```
  SELECT count_if(secondary_mobile NULL) FROM personal_details
  ```
- ```
  SELECT count_if(secondary_mobile IS NULL) FROM personal_details
  ```
  (Correct)
- ```
  SELECT count_null(secondary_mobile) FROM personal_details
  ```
- ```
  SELECT count_if(NULL secondary_mobile) FROM personal_details
  ```

Explanation

To count the number of **NULL** values in a column, various methods can be adopted but the one which is used the most is `count_if()` function. `count_if()` function accepts conditions like `email IS NOT NULL` or `length(phone_number) != 10` etc.

In respect to the above
  ```

Question, the correct condition would be `secondary_mobile IS NULL` which is used in **option C**. As you can see, `count_if()` combines the functionality of `count()` function and `WHERE` clause, the required result can also be achieved using the following query:

```
1. SELECT count(secondary_mobile)
2. FROM personal_details
3. WHERE secondary_mobile IS NULL
```

More Info: Using count_if() function in SQL

Question 22

Which of the following keywords should be used to create a UDF in SQL?

- **UDF**
- **FUNC**
- **FUNCTION**

  (Correct)
- **USER DEFINED FUNCTION**
- **DEF**

Explanation

UDF is an essential part of **Spark SQL** processing. It can be created using the `CREATE FUNCTION` command. It helps in defining **custom logic** that can be applied to different **columns of a table or a view**.

Please note, the UDF command CANNOT be used to create a function.

More Info: Creating UDF in SQL

Question 23

The following SQL statements intend to create a Delta table `company_zones` using a SQLite table named `zones`. Which of the following can replace the blank?

```
1. CREATE TABLE company_zones
2. _______________
3. OPTIONS (
4. url = “jdbc:sqlite:/companyDB”,
5. dbtable = “zones”
6. )
```

- `USING SQLITE`
- `USING DELTA`
- `USING JDBC`

  (Correct)
- `USING DATABASE`
- `USING SQL`

Explanation

For connecting with SQL databases, Databricks uses the `USING JDBC` command. The type of database is identified by the `url` value in the `OPTIONS`. In this

Question, the connection needs to be made with **SQLite database**. Connection made to SQLite database does **not** require **username, password or port number**. For other databases, the `url` value in `OPTIONS` contains the **host name** as well as the **port number** while the **username and password** are also provided in **key value pairs** in `OPTIONS`.

More Info: Querying SQL databases in Databricks

Question 24

The following is a snapshot of the workers table where the `details` column is of array type.

worker_code	details
DB65	[{"name": "Brendon", "age": 23, "salary": 40000}]
AB34	[{"name": "Michael", "age": 27, "salary": 66400}]
...	...

A data engineer needs to create a new column `filtered_workers` which contains values only for those records for which the `salary` of the worker is greater than 10000. They decide to go with higher-order function. Which of the following statements should be used by the data engineer to complete the task?

- ◉ `FILTER (details, i -> i.salary > 10000) AS filtered_workers`

  (Correct)
- ○ `UDF (details, i -> i.salary > 10000) AS filtered_workers`
- ○ `REDUCE (details, i > 10000) AS filtered_workers`
- ○ `FILTER (details, salary > 10000) AS filtered_workers`
- ○ `FILTER (details, i -> i > 10000) AS filtered_workers`

Explanation
Correct SQL statement:

```
FILTER (details, i -> i.salary > 10000) AS filtered_workers
```

`FILTER` is a higher-order function which can filter the contents of an **array column** using a **lambda function**. `FILTER` function takes **two arguments**, the **array column name and the condition**.

Here, `details` is the name of the column and `i -> i.salary > 10000` is the condition. If the condition is **TRUE**, the value is added to the

column `filtered_workers`. Also note, `i` is the variable which iterates over the array which means `i.salary` is the salary of each worker in the array.

More Info: FILTER function in Spark SQL

Question 25

Which of the following is not a feature of AutoLoader?

- **AutoLoader provides incremental processing of new files which are added to cloud storage.**
- **AutoLoader can ingest files from AWS S3 and Google Cloud Storage incrementally**
- **AutoLoader can ingest data in various formats including but not limited to JSON and CSV**
- **AutoLoader uses `_rescued_columns` column for capturing the incompatible data**

  (Correct)
- **AutoLoader converts all the columns to STRING data type when reading from JSON data source**

Explanation
This is a confusing

Question as all the options look correct but one of them is not (partially).
Although **AutoLoader** does use a column for **capturing the incompatible data**, it is not `_rescued_columns` as given in option D. It is rather `_rescued_data` column.
Before moving to the next

Question, I would encourage you to **read all the other options thoroughly** as

Question may arrive in the actual exam asking the correctness of the other options.

More Info: AutoLoader in Databricks

Question 26

Which of the following are the commonly used naming conventions for the three tables of the Incremental multi-hop architecture in Databricks?

- **Bronze -> Silver -> Gold**

  (Correct)

- **Raw -> Silver -> Gold**

- **Silver -> Gold -> Dashboard**

- **Silver -> Gold -> Platinum**

Explanation

In a Databricks **multi-hop architecture**, the most common convention for the table names is **Bronze, Silver and Gold**. This is also called **medallion** architecture since this **coincides** with the **medals** given in a sporting event like Olympics.

More Info: Naming conventions in multi-hop(medallion) architecture

Question 27

A data engineer is working on a PySpark DataFrame `silverDF` as part of a multi-hop architecture. A data analyst needs to perform a one-time query on the DataFrame using SQL in the same session. They have written the following query to make the DataFrame available to the data analyst.

```
silverDF.createOrReplaceTempView('silver_table')
```

Now, the data analyst starts working on the provided temporary view and selects all the data from the view using the following SQL statement but they are not able to view the result.

```
SELECT * FROM silverDF;
```

What could be the reason for non-working of the code?

- **Data engineer has registered the view which cannot be used in SQL as the name of the view and the DataFrame should be identical**
- **Data analyst has used the DataFrame name instead of name of the view**

  (Correct)
- **Data engineer has used wrong function, `createTable()` should have been used**
- **There is no way a PySpark DataFrame can be used in Spark SQL**
- **Data analyst should run the query inside `spark.sql()` function**

Explanation

This is an easy picker. The `SELECT` query is using the **DataFrame** name while the **reference** to the DataFrame has already been created by the name `silver_table`. The correct way of writing this SQL query would be `SELECT * FROM silver_table;`

Also remember, whenever you need to reference a PySpark DataFrame in SQL, you can register the DataFrame as a temporary view

using `createOrReplaceTempView()` operation. You can then run your SQL query using the name of the newly created temporary view.

More Info: Using PySpark DataFrame in SQL

Question 28

A data engineer needs to control the schema for some of the columns while reading the raw JSON data to the Bronze table using AutoLoader. Which of the following is the most efficient way of controlling the schema?

- **Use trigger**
- **It is not possible to define schema**
- **Use `schemaHints`**

  (Correct)
- **Contact the Databricks Administrator**
- **Use `outputMode` as `'append'`**

Explanation

Suppose, you are ingesting **JSON** data to the **Bronze table** and one of the columns contains marks of students containing **only** integer values. Automatically, all the columns are **casted to string**. If you want the `marks` column to be casted as an **integer column** in the Bronze table, you need to append `option("cloudFiles.schemaHints", "marks INT")`.

Remember, if an unsupported data is encountered, the value will be converted to NULL. That is the reason, you should use `schemaHints` only if you are sure about the data type.

More Info: Using schemaHints in AutoLoader

Question 29

A data engineer, who is working on Bronze to Silver hop in medallion architecture, wants to join `performers` and `events` tables on `location` column and retrieve all the records from the `performers` table but only matching records from the `events` table. What type of join can be used to accomplish the task?

The `performers` table contains details like `name`, `genre`, `performer_id` etc. of all the `performers` associated with the company and the `events` table contains details of all the events like `concerts`, `standup shows` etc. and `performer_id` of the performers who performed at the event.

- **LEFT JOIN**
- **RIGHT JOIN**
- **INNER JOIN**
- **OUTER JOIN**
- **Any one from A (LEFT JOIN) and B (RIGHT JOIN)**

  (Correct)

Explanation
As **all** the records from one table and only the **matching** records from the other table need to be retrieved, you can use **LEFT JOIN** as well as **RIGHT JOIN**. If you want to use **LEFT JOIN**, you can keep the `performers` table as **LEFT table** and `events` table as **RIGHT table**. If you want to perform a **RIGHT JOIN**, you can use the `events` table as **LEFT table** and `performers` table as `RIGHT` table.

More Info: Joins in SQL

Question 30

The following Python code block intends to perform the Silver-Gold transition as a part of multi-hop architecture to find out the maximum products from each `country`. The source table is `warehouse` whereas the target table is `max_products`. What should replace the blank to run the code correctly?

```
1. spark.table("warehouse")
2. .groupBy("country")
3. _____________________
4. .write
5. .table("max_products")
```

- .agg(max(products))
- .agg.max.products
- .agg.max("products")
- .agg(max("products"))

  (Correct)
- .max(agg("products"))

Explanation

Correct code block for **Silver-Gold** transition:

```
1. spark.table("warehouse")
2. .groupBy("country")
3. .agg(max("products"))
4. .write
5. .table("max_products")
```

As the

Question demands the maximum number of `products` from each `country`, aggregate function i.e. `agg()` function can be used while `max()` function can be passed as an argument to the `agg()` function.

Also note, Silver to Gold transition always (at least most of the times) involves aggregation.

More Info: Aggregating data in PySpark

Question 31

Which keyword(s) should be added to make a table or a view a Delta Live Table?

- `DELTA`
- `LIVE`

  (Correct)
- `DELTA LIVE`
- `STREAMING`
- **No keyword is required as every table in Databricks is DLT, by default.**

Explanation

This is an easy picker. If you add a `LIVE` keyword before a **table or a view**, you force the system to treat them as a **DLT (Delta Live Table)**.

More Info: Delta Live Tables in Databricks

Question 32

Which of the following about checkpointing is true for a streaming job in Databricks?

- **Checkpointing is used for checking the faulty data in a table**
- **Checkpointing helps in making a job fault tolerant**

  (Correct)
- **Checkpointing can increase the risk of failure**
- **Checkpointing helps in parallel processing of jobs**
- **Checkpointing is used for scheduling a job in CRON syntax**

Explanation

Checkpointing helps in making a job **fault tolerant** by creating **checkpoints** and **restarting** the job from the checkpoints in case of **failure**. It is an essential feature for **streaming jobs** which is why Databricks **encourages** you to enable **checkpointing** if you want your job to be **fault tolerant**.

More Info: Advantages of using checkpointing in Structured Streaming queries

Question 33

Which of the following hops is known for transitioning timestamps into a human-readable format?

- **Raw to Bronze**
- **Bronze to Silver**

  (Correct)
- **Silver to Gold**
- **Raw to Gold**
- **Gold to Silver**

Explanation
As part of **Silver table enrichments**, **timestamps** are converted to **human-readable** format. So, the correct hop from the above options is **Bronze to Silver** hop.

Let us also discuss what are the main functionalities of other hops in multi-hop architecture.

Raw to Bronze – Apply **schema** to the raw data.

Bronze to Silver – Conversion of **timestamps** into **human-readable** format and performing **joins** with other tables.

Silver to Gold – Performing **aggregations**.

More Info: Medallion architecture in Databricks

Question 34

A data engineer gets a lot of files in an AWS S3 directory specified by a Python variable `loc`. The frequency of files is not uniform. Someday only 3-4 files are received while some days it is more than 100. They need to load the new data received into the `users` table every hour. Which of the following is the most efficient and time-saving technique?

- **Use AutoLoader with default `processingTime` to trigger**
- **Use AutoLoader with `processingTime` as 60 minutes to trigger the job**

  (Correct)
- **Manually run the job every hour**
- **Use AutoLoader with Multi-Hop Architecture**
- **Use AutoRun to run the job every 60 minutes**

Explanation

Let us examine all the options one by one.

Use AutoLoader with default `processingTime` to trigger

INCORRECT. Although **AutoLoader** is a correct choice for this scenario but default processing time is **5 seconds**. According to the

Question, you need to load the data **every hour**. So, this will **not** be an efficient solution (as asked in the

Question).

Use AutoLoader with `processingTime` *as 60 minutes to trigger the job*

CORRECT. Using **AutoLoader** with custom processing time of **60 minutes** is the most **efficient** and **time saving** option.

Manually run the job every hour

INCORRECT. This is an alternative solution but the

Question demands most **efficient** and **time saving** technique.

Use AutoLoader with Multi-Hop Architecture

INCORRECT. Multi-Hop architecture is used for **incremental** data processing and **not** to run a job at specified intervals of time.

Use AutoRun to run the job every 60 minutes

INCORRECT. There is **no** AutoRun command or feature in Databricks.

More Info: Using processingTime with AutoLoader in Databricks

Question 35

Which of the following is not one of the features of SQL endpoint?

- **SQL endpoint supports multiple cluster size options**
- **SQL endpoint comes with an Auto-stop feature which can be used to shut down the endpoint after specified time**
- **SQL endpoint can be scaled to multiple clusters**
- **SQL endpoint can be used to run Java applications while preserving the SQL queries**

  (Correct)
- **SQL endpoint can be connected with tools like Tableau and Power BI using the connection details provided by the Databricks for each SQL endpoint**

Explanation

With a recent update from Databricks, SQL endpoint is now known as SQL warehouse.

This

Question tests your in-depth knowledge on SQL endpoints(warehouses). SQL endpoints(warehouses) have a lot of features but you **cannot** use them to run **Java applications**. A SQL endpoint(warehouse) is specially built to run **SQL queries**.

I will encourage you to **read all the other options** as they are some of the features of an SQL endpoint(warehouse). These can come handy in the actual exam.

More Info: SQL warehouse(formerly known as SQL endpoint) in Databricks

Question 36

Which of the following correctly depicts the relationship between a job and a task in Databricks Workflow?

- **A task consists of one or more jobs which can run linearly or in parallel**
- **Job and task are always equal in any Databricks Workflow**
- **A job can consist of number of tasks**

  (Correct)
- **Number of jobs in a Workflow is always greater than the number of tasks**
- **There is no relationship between a job and a task in Databricks Workflow as they are independent of each other.**

Explanation

Let us look at each option one by one.

A task consists of one or more jobs which can run linearly or in parallel

INCORRECT. It is just the **opposite** of that. A **job consists** of single or multiple **tasks**.

Job and task are always equal in any Databricks Workflow

INCORRECT. This is a tricky one as the **number of jobs and tasks** can be **equal** but **not** always. Mostly, a **single job** contains **several tasks** which run one by one or parallelly as per the logic.

A job can consist of number of tasks

CORRECT. While creating a **job**, you can add a **number of tasks** to it. The type of tasks includes **Databricks notebooks, JAR files, Python scripts** etc.

Number of jobs in a Workflow is always greater than the number of tasks

INCORRECT. A **single job** contains a **number of tasks** as part of a Workflow.

There is no relationship between a job and a task in Databricks Workflow as they are independent of each other.

INCORRECT. Job and task are **not** independent to each other as a **task is a part of a job**. You can think of a **job as a box** and **tasks as balls** in that box.

More Info: Workflows with jobs and tasks in Databricks

Question 37

A team of data analysts is working on a DLT pipeline using SQL which updates the real time weather conditions in different parts of the country. One of the data analysts need to look at the quality of the data loaded to the system. They need to drop the row which has NULL value in `temperature` column. Which of the following SQL statements should be added by them to accomplish this task?

- ```
  CONSTRAINT temp_in_range EXPECT (temperature is NOT NULL)
  ```

- ```
  CONSTRAINT temp_in_range EXPECT (temperature is NOT NULL) ON VIOLATION DROP ROW
  ```

  (Correct)

- ```
  CONSTRAINT temp_in_range EXPECT (temperature is NOT NULL) ON VIOLATION FAIL UPDATE
  ```

- ```
  CONSTRAINT temp_in_range EXPECT (temperature is NULL) ON VIOLATION DROP ROW
  ```

- ```
  CONSTRAINT temp_in_range EXPECT (temperature is NOT NULL) ON VIOLATION DELETE ROW
  ```

Explanation
Before answering this
  ```

Question let us first understand what are the different ways to handle the violations while working with **DLT using SQL**. There are **three** types of **violation handling**.

1. Failing the pipeline altogether once a violation is encountered

2. Dropping the record causing violation and proceeding with the next record

3. Reporting the violations in metrics while adding that record to the target table

Let us see the **syntax** for **all three**:

1. `CONSTARINT constraint_name EXPECT (condition) ON VILOATION FAIL UPDATE` – As soon as the condition is **not** met, the pipeline will **fail**
2. `CONSTARINT constraint_name EXPECT (condition) ON VILOATION DROP ROW` – If the condition is **not** met, the **record will be dropped**
3. `CONSTARINT constraint_name EXPECT (condition)` – The **row will be added to the target table** but the record will be shown under `failed_records` in the **Data Quality dashboard**.

As the

Question states that you need to **drop** the row violating the `NOT NULL` constraint of the `temperature` column, adding `CONSTRAINT temp_in_range EXPECT (temperature is NOT NULL) ON VIOLATION DROP ROW` will get the desired results.

More Info: Data quality constraints in Delta Live Tables

Question 38

Which of the following about the cron scheduling of jobs in Databricks is correct?

- **Cron scheduling can be used for Manual schedule type**
- **Cron scheduling can be used for creating a new cluster**
- **Cron scheduling should be set in Data tab from left side menu**
- **Cron scheduler supports time zone selection**

  (Correct)
- **Cron scheduled jobs cannot be edited**

Explanation

Let us look at each statement one by one.

Cron scheduling can be used for Manual schedule type

INCORRECT. For **scheduling a job**, **Scheduled** is to be selected in **Schedule Type** tab.

Cron scheduling can be used for creating a new cluster

INCORRECT. cron scheduling is used to **schedule a job**. New clusters **cannot** be created using cron syntax.

Cron scheduling should be set in Data tab from left side menu

INCORRECT. For using cron scheduler, you need to go to **Jobs tab** from left side menu.

Cron scheduler supports time zone selection

CORRECT. For scheduling a job, **starting time**, **time interval** and **time zone** can be selected.

Cron scheduled jobs cannot be edited

INCORRECT. You can always **edit the job** even if it is scheduled using **cron job scheduler**.

More Info: Cron job scheduling in Databricks

Question 39

As a data engineer you need to create a football match scorecard for all the users who log into your website. The team has decided to go with the DLT pipeline and put arguments on which type of pipeline to be used for this project i.e. continuous or triggered pipeline. Which of the following arguments made by the members of your team is incorrect?

- **Member A states - A continuous pipeline assures that the end users will get the latest score as they log in to the website**
- **Member B says - By using a continuous pipeline our cost will decrease**

  (Correct)
- **Member C thinks - The triggered pipeline will make the score refresh rate very slow**
- **Member D writes - For live events like a soccer match we should use a continuous pipeline**
- **Member E proposes – Triggered pipelines can be scheduled or run manually**

Explanation

It seems like **member B** has **not** done their research properly about the **DLT pipelines**. As you (*should*) know DLT pipelines can be categorized in two parts based on their **continuity or refresh rate**.

First one being the **triggered pipeline** and other one the **continuous pipeline**.

The **continuous** pipeline, as the name suggests, is better suited for **real time data** (such as a **live event** sending some amount of data) and requires a **running cluster** which makes this a **costly** option.

On the other hand, the **triggered** pipeline fetches the data from the source as soon as it is **triggered**, and the cluster runs **only** till the pipeline is being executed. This makes clear that **member B is not correct** since by using a continuous pipeline you tend to incur **more** compute costs (but the **performance** will **increase**).

More Info: Triggered vs Continuous pipelines in Databricks

Question 40

An organization wants to decrease their cost of SQL endpoint which is currently used by only one of their data analysts to query the data once or twice daily. The queries are not complex and can afford latency. What all steps can be taken in order to decrease the cost incurred?

- **Select the minimum cluster size**
- **Turn on the Auto-stop feature**
- **Select minimum values for Scaling the endpoint**
- **All of the above can help in reducing the cost**

  (Correct)
- **None of the above**

Explanation

Let us look at all the options and try to understand which of these can actually help us in saving the costs.

Select the minimum cluster size

The cost of an endpoint is measured in **DBU/hour**. Thus, decreasing the cluster size will help in decreasing the cost as the number of DBUs are proportional to the cluster size.

Turn on the Auto-stop feature

The cost of a SQL endpoint also depends on the **number of hours** it remains active. In order to **decrease the cost**, you can turn on the **Auto-stop feature** which automatically turns off the endpoint after the specified minutes of **inactivity**.

Select minimum values for Scaling the endpoint

By selecting minimum values for scaling, the **number of DBUs decreases** which, in turn, decreases the overall cost.

Hence, option D is correct as all these techniques help in reducing the cost.

As per the latest changes from Databricks, *SQL endpoint* has been renamed to *SQL warehouse*.

More Info: Reducing cost of a SQL endpoint(now known as SQL warehouse)

Question 41

Which of the following can be called as the unit of computation cost while creating a DLT pipeline?

- **DPU/hour**
- **DPU/second**
- **DBU/hour**

  (Correct)
- **Number of seconds**
- **Number of hours**

Explanation
DBU stands for **Databricks Unit**. The number of **Databricks units** consumed **per hour** is the **total cost** of your **DLT pipeline**.

More Info:

Question 42

Your colleague needs to provide select and metadata read permissions on database `app_test` to a user named `abc`. They have written the following query to grant the permission.

```
GRANT SELECT, READ_METADATA TO DATABASE app_test ON `abc`
```

What should be changed in the above statement to make this work?

- `GRANT` **should be replaced with** `GRANTS` **as there are multiple grants in one statement**
- `READ_METADATA` **should be omitted as all the users have metadata read permission**
- `READ_METADATA` **should be omitted as granting** `SELECT` **permission implies that the user can read the metadata**
- **The position of** `ON` **and** `TO` **should be swapped**

  (Correct)
- `DATABASE` **keyword should be replaced by** `CATALOG`

Explanation
Correct code block:

```
GRANT SELECT, READ_METADATA ON DATABASE app_test TO `abc`
```

The query seems to be correct but the **positions of** `TO` **and** `ON` should be **swapped**. `ON DATABASE` comes first with the **database name** which is followed by `TO` while specifying the **name of the user or group** to which the grant needs to be issued. Also note, `GRANTS` is also a valid keyword but is used to **view the grants** on a specific data object like **database**, **table** etc.

More Info: GRANT syntax to grant permissions to users and groups in Databricks

Question 43

A data engineer has created a `request` table. Another data engineer has joined the team and needs all the privileges to that table. The data engineer does not remember the SQL command and wants to use the UI to grant permissions. Which of the following can be used by them to accomplish the task?

- **AutoLoader**
- **Data Explorer**

  (Correct)
- **Git**
- **Multi-Hop architecture**
- **Data Lakehouse**

Explanation

Data Explorer can be used for all the **permissions** related tasks. Data Explorer can be used for various activities like **managing ownership and permissions of databases and tables**, **viewing schema and properties of the tables and the databases** etc.

More Info: Using Data Explorer for managing permissions in Databricks

Question 44

Which of the following is true for the features of Unity Catalog in Databricks?

- **Unity catalog is based on ANSI SQL**
- **It allows fine-grain access to specific rows and column**
- **Unity catalog can be used to govern data on different clouds**
- **Unity catalog can control your existing catalogs**
- **All of the above**

  (Correct)

Explanation
All the above options describe the features of **Unity Catalog**. I will encourage you to **read all the options** as you may experience a

Question on Unity Catalog in the actual exam as well. Knowing these features will help you in answering the

Question involving Unit Catalog with confidence.

More Info: Advantages of using Unity Catalog

Question 45

Which of the following SQL statements can be used by the Databricks admin to change the owner of database `error_logs` to user `dan@nad.adn`

- ```
  ALTER DATABASE error_logs OWNERSHIP TO `dan@nad.adn`
  ```
- ```
  GRANT OWNER FOR SCHEMA error_logs TO `dan@nad.adn`
  ```
- ```
  CHANGE OWNER TO `dan@nad.adn` FOR SCHEMA error_logs
  ```
- ```
  ALTER DATABSE error_logs GRANT OWNER TO `dan@nad.adn`
  ```
- ```
  ALTER SCHEMA error_logs OWNER TO `dan@nad.adn`
  ```

  (Correct)

Explanation
Before answering this
  ```

Question, you should know **who can transfer the ownership of data objects** like tables and databases. The transfer of ownership can be carried by the **current owner** of the object or the **Databricks admin** only. The following is the syntax for assigning a new owner to a database(schema):

```
ALTER SCHEMA schema-name OWNER TO `user-name`
```

Also note, `SCHEMA` and `DATABASE` can be used **interchangeably**.

More Info: Transferring ownership of a data object