PassQueen
a.k.a dumpsqueen.com

SnowPro Advanced Administrator

Snowflake ADA-C01

Version Demo

Total Demo Questions: 9

Total Premium Questions: 98

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Account and Security	37
Topic 2, Data Governance	8
Topic 3, Monitoring and Performance	22
Topic 4, Business Continuity and Disaster Recovery	14
Topic 5, Data Sharing	15
Topic 6, Mix Questions	2
Total	98

QUESTION NO: 1

A resource monitor named MONTHLY_FINANCE_LIMIT has been created and applied to two virtual warehouses (fin_wh1 and fin_wh2) using the following SQL:

```
ALTER RESOURCE MONITOR MONTHLY_FINANCE_LIMIT SET CREDIT_QUOTA = 1000
    FREQUENCY = MONTHLY
    START_TIMESTAMP = '2022-12-01 00:00 PST'
    NOTIFY_USERS_ON_80_PERCENT DO SUSPEND
    NOTIFY_USERS_ON_100_PERCENT DO SUSPEND_IMMEDIATE;

ALTER WAREHOUSE fin_wh1 SET RESOURCE_MONITOR = MONTHLY_FINANCE_LIMIT;
ALTER WAREHOUSE fin_wh2 SET RESOURCE_MONITOR = MONTHLY_FINANCE_LIMIT;
```

Given that the combined total of credits consumed by fin_wh1 and fin_wh2 (including cloud services) has reached 800 credits and both warehouses are suspended, what should the ACCOUNTADMIN execute to allow both warehouses to be resumed? (Select TWO).

A. ALTER WAREHOUSE fin_wh1 RESUME;

B. ALTER WAREHOUSE fin_wh2 RESUME;

C. ALTER WAREHOUSE fin_wh1 UNSET RESOURCE_MONITOR MONTHLY_FINANCE_LIMIT;

D. ALTER WAREHOUSE fin_wh2 UNSET RESOURCE_MONITOR MONTHLY_FINANCE_LIMIT;

E. ALTER RESOURCE MONITOR MONTHLY_FINANCE_LIMIT SET CREDIT_QUOTA = 1500;

F. ALTER RESOURCE MONITOR MONTHLY_FINANCE_LIMIT RESET;

G. ALTER WAREHOUSE fin_wh1 UNSET RESOURCE_MONITORS;

ANSWER: E F

Explanation:

`ALTER RESOURCE MONITOR MONTHLY_FINANCE_LIMIT SET CREDIT_QUOTA = 1500;` is correct because it increases the monitor quota while the recorded consumption remains at 800 credits. With a 1,500-credit quota, that consumption is below the configured 80% suspension threshold. Snowflake permits a warehouse suspended by a resource monitor to be resumed after the monitor's quota is increased sufficiently to remove the condition that caused the suspension. The warehouses still need to be resumed separately after the monitor is no longer enforcing that suspension state.

`ALTER RESOURCE MONITOR MONTHLY_FINANCE_LIMIT RESET;` is also correct because it resets the resource monitor's current credit-usage accounting for its monitoring interval. The monitor therefore no longer has usage at its suspension threshold, which permits the attached warehouses to be resumed. Resetting is useful when an administrator intentionally starts a fresh monitoring interval rather than changing the established quota. Both commands require the appropriate authority to modify the resource monitor, such as ACCOUNTADMIN ownership or a role granted the necessary privileges. Snowflake documents both changing a credit quota and resetting a monitor in the ALTER RESOURCE MONITOR reference; the behavior of monitor-triggered warehouse suspension is covered in the Resource monitors documentation.

QUESTION NO: 2

The following SQL command was executed:

Use role SECURITYADMIN;

Grant ownership

On future tables

In schema PROD. WORKING

To role PROD_WORKING_OWNER;

Grant role PROD_WORKING_OWNER to role SYSADMIN;

Use role ACCOUNTADMIN;

Create table PROD.WORKING.XYZ (value number) ;

Which role(s) can alter or drop table XYZ?

A. Because ACCOUNTADMIN created the table, only the ACCOUNTADMIN role can alter or drop table XYZ.

B. SECURITYADMIN, SYSADMIN, and ACCOUNTADMIN can alter or drop table XYZ.

C. PROD_WORKING_OWNER, ACCOUNTADMIN, and SYSADMIN can alter or drop table XYZ.

D. Only the PROD_WORKING_OWNER role can alter or drop table XYZ.

ANSWER: C

Explanation:

`PROD_WORKING_OWNER, ACCOUNTADMIN, and SYSADMIN can alter or drop table XYZ`. The future ownership grant applies to tables subsequently created in the `PROD.WORKING` schema. Consequently, although `ACCOUNTADMIN` executes the `CREATE TABLE` statement, ownership of the newly created `XYZ` table is assigned to `PROD_WORKING_OWNER` under the future grant. Ownership is the privilege that provides full control of an object, including the authority to alter and drop it.

`SYSADMIN` can also perform these operations because `PROD_WORKING_OWNER` was granted to `SYSADMIN`. Snowflake role hierarchy is inherited upward: a role that is granted another role inherits that granted role's object privileges when it is active. Therefore, `SYSADMIN` inherits the table ownership capability held by `PROD_WORKING_OWNER`. Finally, `ACCOUNTADMIN` inherits the `SYSADMIN` role in Snowflake's system role hierarchy, and thus also inherits this capability. Snowflake documents future ownership grants and their effect on objects created later in a schema at GRANT OWNERSHIP. The inheritance behavior of roles, including the system role hierarchy, is described in Role hierarchy and privilege inheritance.

QUESTION NO: 3

An Administrator is designing a database role model for the SALES database. Which statements describe valid uses of database roles? (Select TWO).

A. Grant a database role to an account role.

B. Grant a database role to another database role in the same database.

C. Grant a database role directly to a user.

D. Grant an account role to a database role.

E. Grant a database role to a database role in a different database.

ANSWER: A B

Explanation:

A database role can be granted to an account role. This allows an account role to inherit the database privileges encapsulated by the database role and is the usual way to make database-scoped privileges available to users.

A database role can also be granted to another database role in the same database, supporting a hierarchy of database-specific roles. Database roles cannot be granted directly to users, and an account role cannot be granted to a database role. These restrictions preserve the database-scoped nature of database roles.

QUESTION NO: 4

An Administrator receives data from a Snowflake partner. The partner is sharing a dataset that contains multiple secure views. The Administrator would like to configure the

data so that only certain roles can see certain secure views.

How can this be accomplished?

A. Apply RBAC directly onto the partner's shared secure views.

B. Individually grant imported privileges onto the schema in the share.

C. Clone the data and insert it into a company-owned share and apply the desired RBAC on the new tables.

D. Create views over the incoming shared database and apply the desired RBAC onto these views.

ANSWER: D

Explanation:

Creating views over the incoming shared database and applying the desired RBAC onto these views is the appropriate approach because the consumer account can create its own local, secure views that reference objects provided through the partner's shared database. These locally owned views are governed by the consumer account's own access-control model. The administrator can therefore grant the required database, schema, and view privileges to selected account roles, allowing each role to query only the local views it is authorized to use.

This pattern provides a controlled presentation layer over the shared dataset. A local secure view can expose only the required columns, rows, or transformations from a partner-provided secure view, while Snowflake RBAC determines which consumer roles can use that local view. The partner continues to own and manage the shared data and original secure views, while the consumer independently manages access for its users without copying the shared data. Snowflake documents that consumers can create secure views over shared data, and that privileges on the consumer-owned views can be granted to roles in the consumer account. See Creating Views on Shared Data and Access Control Overview.

QUESTION NO: 5

Which command can temporarily disable Multi-factor Authentication (MFA) for the Snowflake username user1 for 24 hours?

A. alter user user1 set MINS_TO_BYPASS_MFA=1440;

B. alter user userl set DISABLE_MFA=1440;

C. alter user userl set TEMPORARY_MFA_BYPASS=1440;

D. alter user userl set HOURS_TO_BYPASS_MFA=24;

ANSWER: A

Explanation:

`ALTER USER user1 SET MINS_TO_BYPASS_MFA = 1440;` is correct because `MINS_TO_BYPASS_MFA` is the Snowflake user property designed to create a temporary MFA bypass for an individual user. Its value is expressed in minutes. Twenty-four hours contains 1,440 minutes, so setting the property to `1440` permits the user to sign in without completing the configured MFA challenge during that temporary period. This setting is useful when an administrator must restore access for a user who cannot use their MFA authenticator, while retaining the normal MFA configuration rather than removing it. Once the configured bypass interval expires, Snowflake again requires MFA for subsequent authentication

attempts. Administrators can also end an active bypass before it expires by setting `MINS_TO_BYPASS_MFA` to `0`. The command is executed by a role with the required authority to alter the user, such as a role granted ownership of that user or an appropriate user-administration role. Snowflake documents this property in the ALTER USER syntax and describes temporary MFA bypass behavior in its MFA administration guidance. See Snowflake ALTER USER documentation and Snowflake MFA documentation.

QUESTION NO: 6

MY_TABLE is a table that has not been updated or modified for several days. On 01 January 2021 at 07:01, a user executed a query to update this table. The query ID is

'8e5d0ca9-005e-44e6-b858-a8f5b37c5726'. It is now 07:30 on the same day.

Which queries will allow the user to view the historical data that was in the table before this query was executed? (Select THREE).

A. SELECT * FROM my table WITH TIME_TRAVEL (OFFSET => -60*30);

B. SELECT * FROM my_table AT (TIMESTAMP => '2021-01-01 07:00:00' :: timestamp);

C. SELECT * FROM TIME_TRAVEL ('MY_TABLE', 2021-01-01 07:00:00);

D. SELECT * FROM my table PRIOR TO STATEMENT '8e5d0ca9-005e-44e6-b858-a8f5b37c5726';

E. SELECT * FROM my_table AT (OFFSET => -60*30);

F. SELECT * FROM my_table BEFORE (STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');

ANSWER: B E F

Explanation:

Snowflake Time Travel supports querying a table as it existed at a specified point before a data-changing operation. `SELECT * FROM my_table AT (TIMESTAMP => '2021-01-01 07:00:00' :: timestamp);` selects the version at 07:00, which precedes the update at 07:01. The timestamp form of the `AT` clause is appropriate when the desired historical instant is known explicitly.

`SELECT * FROM my_table AT (OFFSET => -60*30);` also resolves to 07:00 when evaluated at 07:30: an offset is expressed in seconds relative to the current time, and negative values refer to the past. Therefore, this query accesses the table state thirty minutes earlier, before the update.

`SELECT * FROM my_table BEFORE (STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');` uses the update statement's query ID as the Time Travel reference. `BEFORE` is exclusive, so it returns the table version immediately before that statement's effects were applied. Snowflake documents the `AT | BEFORE` clause and its TIMESTAMP, OFFSET, and STATEMENT parameters at AT | BEFORE. Additional Time Travel behavior and retention details are available in the Snowflake Time Travel documentation.

QUESTION NO: 7

A Snowflake account is configured with SCIM provisioning for user accounts and has bi-directional synchronization for user identities. An Administrator with access to SECURITYADMIN uses the Snowflake UI to create a user by issuing the following commands:

use role USERADMIN;

create or replace role DEVELOPER_ROLE;

create user PTORRES PASSWORD = 'hello world!' MUST_CHANGE_PASSWORD = FALSE

default_role = DEVELOPER_ROLE;

The new user named PTORRES successfully logs in, but sees a default role of PUBLIC in the web UI. When attempted, the following command fails:

use DEVELOPER_ROLE;

Why does this command fail?

A. The DEVELOPER_ROLE needs to be granted to SYSADMIN before user PTORRES will be able to use the role.

B. The new role can only take effect after USERADMIN has logged out.

C. USERADMIN needs to explicitly grant the DEVELOPER_ROLE to the new USER.

D. The new role will only take effect once the identity provider has synchronized by way of SCIM with the Snowflake account.

ANSWER: C

Explanation:

USERADMIN needs to explicitly grant the DEVELOPER_ROLE to the new USER because assigning a role as a user's `DEFAULT_ROLE` does not itself establish role membership. In Snowflake's role-based access-control model, a user can activate a role only when that role has been granted directly or indirectly to the user. The `DEFAULT_ROLE` property merely identifies which already-granted role Snowflake should attempt to activate when the user starts a session. It is not a substitute for `GRANT ROLE`.

After creating the role and user, a role owner or an authorized administrator should run `GRANT ROLE DEVELOPER_ROLE TO USER PTORRES;`. Once that grant is in place, PTORRES has membership in the role, can execute `USE ROLE DEVELOPER_ROLE`, and can receive it as the default role at login. Snowflake documents that a role specified as `DEFAULT_ROLE` must be granted to the user, and documents the required syntax for granting account roles to users. SCIM identity synchronization does not remove this core Snowflake authorization requirement for a Snowflake-managed role assignment.

References: Snowflake CREATE USER reference and Snowflake GRANT ROLE reference.

QUESTION NO: 8

What are characteristics of data replication in Snowflake? (Select THREE).

A. The ALTER DATABASE ... ENABLE REPLICATION TO ACCOUNTS command must be issued from the primary account.

B. Users must be granted REPLICATIONADMIN privileges in order to enable replication.

C. To start replication run the ALTER DATABASE ... REFRESH command on the account where the secondary database resides.

D. Replication can only occur within the same cloud provider.

E. Databases created from shares can be replicated.

F. Users can have unlimited primary databases and they can be replicated to an unlimited number of accounts if all accounts are within the same organization.

G. Replication can occur across different cloud providers and regions when supported by Snowflake.

ANSWER: A C G

Explanation:

Database replication is configured from the source, or primary, database. Therefore, the command `ALTER DATABASE ... ENABLE REPLICATION TO ACCOUNTS` is issued while using the primary account that owns the primary database. This

operation establishes the specified accounts as permitted destinations for secondary copies. A role performing this operation must have the required authority on the primary database.

After replication has been enabled and a secondary database has been created in a target account, refresh activity is initiated in the account containing that secondary database. Running `ALTER DATABASE ... REFRESH` there synchronizes the secondary database with the latest replicated state of its primary database. This is the mechanism used to update a secondary copy outside of replication-group scheduling workflows.

Snowflake replication also supports replication between accounts in different regions and, where supported by Snowflake, across cloud platforms. The participating accounts must belong to the same organization, and the relevant replication prerequisites and regional/cloud availability requirements apply. This capability enables disaster-recovery and business-continuity designs that do not rely solely on a single region or cloud provider. See Snowflake's database replication introduction and the ALTER DATABASE reference for command and replication details.

QUESTION NO: 9

A Snowflake organization MYORG consists of two Snowflake accounts:

The ACCOUNT1 has a database PROD_DB and the ORGADMIN role enabled.

Management wants to have the PROD_DB database replicated to ACCOUNT2.

Are there any necessary configuration steps in ACCOUNT1 before the database replication can be configured and initiated in ACCOUNT2?

A. USE ROLE ORGADMIN;SELECT SYSTEMSGLOBAL_ACCOUNT_SET_PARAMETER ('MYORG. ACCOUNT1', 'ENABLE_ACCOUNT_DATABASE_REPLICATION', 'TRUE');SELECT SYSTEMSGLOBAL_ACCOUNT_SET_PARAMETER ('MYORG. ACCOUNT2', 'ENABLE_ACCOUNT_DATABASE_REPLICATION', 'TRUE');USE ROLE ACCOUNTADMIN;ALTER DATABASE PROD DB ENABLE REPLICATION TO ACCOUNTS MYORG. ACCOUNT2;

B. USE ROLE ORGADMIN;SELECT SYSTEM$GLOBAL_ACCOUNT_SET_PARAMETER('MYORG.ACCOUNT1', 'ENABLE_ACCOUNT_DATABASE_REPLICATION', 'TRUE');USE ROLE ACCOUNTADMIN;ALTER DATABASE PROD_DB ENABLE REPLICATION TO ACCOUNTS MYORG.ACCOUNT2 IGNORE EDITION CHECK;

C. No configuration steps are necessary in ACCOUNT1. Replicating databases across accounts within the same Snowflake organization is enabled by default.

D. It is not possible to replicate a database from an Enterprise edition Snowflake account to a Standard edition Snowflake account.

ANSWER: B

Explanation:

`USE ROLE ORGADMIN;` followed by `SYSTEM$GLOBAL_ACCOUNT_SET_PARAMETER` for `MYORG.ACCOUNT1`, and then enabling replication on `PROD_DB`, is the required source-account preparation. The organization administrator enables the `ENABLE_ACCOUNT_DATABASE_REPLICATION` parameter for the account that contains the primary database. This organization-level operation authorizes ACCOUNT1 to act as a source for database replication. Once that authorization is in place, an account administrator in ACCOUNT1 can execute `ALTER DATABASE PROD_DB ENABLE REPLICATION TO ACCOUNTS MYORG.ACCOUNT2` to designate ACCOUNT2 as a replication target. The target account can then create and refresh a secondary database from the primary database's replication configuration. The `IGNORE EDITION CHECK` clause permits the configuration to proceed when the target account is on a lower Snowflake edition than the source, where applicable. Snowflake documents the source-account enablement and the process for establishing primary and secondary databases in its database replication overview. The syntax and account-target configuration for the source database are documented in the ALTER DATABASE reference.