Data Build Tool (DBT) Interview Questions and Answers

nishad patkar · Follow
11 min read · Dec 21, 2023

43 1

Data Build Tool (DBT) is a popular open-source tool used in the data analytics and data engineering fields. DBT helps data professionals transform, model, and prepare data for analysis. If you're preparing for an interview related to DBT, it's important to be well-versed in its concepts and functionalities. To help you prepare, here's a list of common interview questions and answers about DBT.

1. What is DBT?

Answer: DBT, short for Data Build Tool, is an open-source data transformation and modeling tool. It helps analysts and data engineers manage the transformation and preparation of data for analytics and reporting.

2. What are the primary use cases of DBT?

Answer:DBT is primarily used for data transformation, modeling, and preparing data for analysis and reporting. It is commonly used in data warehouses to create and maintain data pipelines.

3. How does DBT differ from traditional ETL tools?

Answer: Unlike traditional ETL tools, DBT focuses on transforming and modeling data within the data warehouse itself, making it more suitable for ELT (Extract, Load, Transform) workflows. DBT leverages the power and scalability of modern data warehouses and allows for version control and testing of data models.

4. What is a DBT model?

Answer: A DBT model is a SQL file that defines a transformation or a table within the data warehouse. Models can be simple SQL queries or complex transformations that create derived datasets.

5. Explain the difference between source and model in DBT.

Answer: A source in DBT refers to the raw or untransformed data that is ingested into the data warehouse. Models are the transformed and structured datasets created using DBT to support analytics.

6. What is a DBT project?

Answer: A DBT project is a directory containing all the files and configurations necessary to define data models, tests, and documentation. It is the primary unit of organization for DBT.

7. What is a DAG in the context of DBT?

Answer: DAG stands for Directed Acyclic Graph, and in the context of DBT, it represents the dependencies between models. DBT uses a DAG to determine the order in which models are built.

8. How do you write a DBT model to transform data?

Answer: To write a DBT model, you create a `.sql` file in the appropriate project directory, defining the SQL transformation necessary to generate the target dataset.

9. What are DBT macros, and how are they useful in transformations?

Answer: DBT macros are reusable SQL code snippets that can simplify and standardize common operations in your DBT models, such as filtering, aggregating, or renaming columns.

10. How can you perform testing and validation of DBT models?

Answer: You can perform testing in DBT by writing custom SQL tests to validate your data models. These tests can check for data quality, consistency, and other criteria to ensure your models are correct.

11. Explain the process of deploying DBT models to production.

Answer: Deploying DBT models to production typically involves using DBT Cloud, CI/CD pipelines, or other orchestration tools. You'll need to compile and build the models and then deploy them to your data warehouse environment.

12. How does DBT support version control and collaboration?

Answer: DBT integrates with version control systems like Git, allowing teams to collaborate on DBT projects and track changes to models over time. It provides a clear history of changes and enables collaboration in a multi-user environment.

13. What are some common performance optimization techniques for DBT models?

Answer: Performance optimization in DBT can be achieved by using techniques like materialized views, optimizing SQL queries, and using caching to reduce query execution times.

14. How do you monitor and troubleshoot issues in DBT?

Answer: DBT provides logs and diagnostics to help monitor and troubleshoot issues. You can also use data warehouse-specific monitoring tools to identify and address performance problems.

15. Can DBT work with different data sources and data warehouses?

Answer: Yes, DBT supports integration with a variety of data sources and data warehouses, including Snowflake, BigQuery, Redshift, and more. It's adaptable to different cloud and on-premises environments.

16. How does DBT handle incremental loading of data from source systems?

Answer: DBT can handle incremental loading by using source freshness checks and managing data updates from source systems. It can be configured to only transform new or changed data.

17. What security measures does DBT support for data access and transformation?

Answer: DBT supports the security features provided by your data warehouse, such as row-level security and access control policies. It's important to implement proper access controls at the database level.

18. How can you manage sensitive data in DBT models?

Answer: Sensitive data in DBT models should be handled according to your organization's data security policies. This can involve encryption, tokenization, or other data protection measures.

19. Types of Materialization?

Answer: DBT supports several types of materialization are as follows:

1)View (Default):

Purpose: Views are virtual tables that are not materialized. They are essentially saved queries that are executed at runtime.
Use Case: Useful for simple transformations or when you want to reference a SQL query in multiple models.

```
{{ config(
  materialized='view'
) }}
SELECT
  ...
FROM ...
```

2)Table:

Purpose: Materializes the result of a SQL query as a physical table in your data warehouse.
Use Case: Suitable for intermediate or final tables that you want to persist in your data warehouse.

```
{{ config(
  materialized='table'
) }}
SELECT
  ...
INTO {{ ref('my_table') }}
FROM ...
```

3)Incremental:

data loads.
Use Case: Ideal for situations where you want to update your table with only the new or changed data since the last run.

```
{{ config(
  materialized='incremental'
) }}
SELECT
  ...
FROM ...
```

4)Table + Unique Key:

Purpose: Similar to the incremental materialization, but specifies a unique key that dbt can use to identify new or updated rows.

Use Case: Useful when dbt needs a way to identify changes in the data.

```
{{ config(
  materialized='table',
  unique_key='id'
) }}
SELECT
  ...
INTO {{ ref('my_table') }}
FROM ...
```

5)Snapshot:

Purpose: Materializes a table in a way that retains a version history of the data, allowing you to query the data as it was at different points in time.

Use Case: Useful for slowly changing dimensions or situations where historical data is important.

```
{{ config(
  materialized='snapshot'
) }}
SELECT
  ...
INTO {{ ref('my_snapshot_table') }}
FROM ...
```

20. Types of Tests in DBT?

Answer: Dbt provides several types of tests that you can use to validate your data. Here are some common test types in dbt:

1)Unique Key Test (***`unique`*** *):*

Verifies that a specified column or set of columns contains unique values.

```
version: 2

models:
  - name: my_model
    tests:
      - unique:
          columns: [id]
```

2)Not Null Test (***`not_null`*** *):*

Ensures that specified columns do not contain null values.

```
version: 2

models:
  - name: my_model
    tests:
      - not_null:
          columns: [name, age]
```

3)Accepted Values Test (`accepted_values`):

Validates that the values in a column are among a specified list.

```
version: 2

models:
  - name: my_model
    tests:
      - accepted_values:
          column: status
          values: ['active', 'inactive']
```

4)Relationship Test (`relationship`):

Verifies that the values in a foreign key column match primary key values in the referenced table.

```
version: 2

models:
  - name: orders
    tests:
      - relationship:
          to: ref('customers')
          field: customer_id
```

5)Referential Integrity Test (`referential integrity`):

Checks that foreign key relationships are maintained between two tables.

```
version: 2

models:
  - name: orders
    tests:
      - referential_integrity:
          to: ref('customers')
          field: customer_id
```

6)Custom SQL Test (`custom_sql`):

Allows you to define custom SQL expressions to test specific conditions.

```
version: 2

models:
  - name: my_model
    tests:
      - custom_sql: "column_name > 0"
```

21.What is seed?

Answer: A "seed" refers to a type of dbt model that represents a table or view containing static or reference data. Seeds are typically used to store data that doesn't change often and doesn't require transformation during the ETL (Extract, Transform, Load) process.

Here are some key points about seeds in dbt:

1. **Static Data:** Seeds are used for static or reference data that doesn't change frequently. Examples include lookup tables, reference data, or any data that serves as a fixed input for analysis.
2. **Initial Data Load:** Seeds are often used to load initial data into a data warehouse or data mart. This data is typically loaded once and then used as a stable reference for reporting and analysis.
3. **YAML Configuration:** In dbt, a seed is defined in a YAML file where you specify the source of the data and the destination table or view in your data warehouse. The YAML file also includes configurations for how the data should be loaded.

Here's an example of a dbt seed YAML file:

```yaml
version: 2

sources:
  - name: my_seed_data
    tables:
      - name: my_seed_table
        seed:
          freshness: { warn_after: '7 days', error_after: '14 days' }
```

22.What is Pre-hook and Post-hook?

Answer: Pre-hooks and Post-hooks are mechanisms to execute SQL commands or scripts before and after the execution of dbt models, respectively. dbt is an open-source tool that enables analytics engineers to transform data in their warehouse more effectively.

Here's a brief explanation of pre-hooks and post-hooks:

1)Pre-hooks:

- A pre-hook is a SQL command or script that is executed before running dbt models.
- It allows you to perform setup tasks or run additional SQL commands before the main dbt modeling process.
- Common use cases for pre-hooks include tasks such as creating temporary tables, loading data into staging tables, or performing any other necessary setup before model execution.

Example of a pre-hook :

```
-- models/my_model.sql
{{ config(
  pre_hook = "CREATE TEMP TABLE my_temp_table AS SELECT * FROM my_source_table"
) }}
SELECT
  column1,
  column2
FROM
  my_temp_table
```

2)Post-hooks:

- A post-hook is a SQL command or script that is executed after the successful completion of dbt models.
- It allows you to perform cleanup tasks, log information, or execute additional SQL commands after the models have been successfully executed.
- Common use cases for post-hooks include tasks such as updating metadata tables, logging information about the run, or deleting temporary tables created during the pre-hook.

Example of a post-hook :

```
-- models/my_model.sql
SELECT
  column1,
  column2
FROM
  my_source_table

{{ config(
  post_hook = "UPDATE metadata_table SET last_run_timestamp = CURRENT_TIMESTAMP"
) }}
```

23.what is snapshots?

Answer: “snapshots” refer to a type of dbt model that is used to track changes over time in a table or view. Snapshots are particularly useful for building historical reporting or analytics, where you want to analyze how data has changed over different points in time.

Here's how snapshots work in dbt:

1. **Snapshot Tables:** A snapshot table is a table that represents a historical state of another table. For example, if you have a table representing customer information, a snapshot table could be used to capture changes to that information over time.
2. **Unique Identifiers:** To track changes over time, dbt relies on unique identifiers (primary keys) in the underlying data. These identifiers are used to determine which rows have changed, and dbt creates new records in the snapshot table accordingly.
3. **Timestamps:** Snapshots also use timestamp columns to determine when each historical version of a record was valid. This allows you to query the data as it existed at a specific point in time.
4. **Configuring Snapshots:** In dbt, you configure snapshots in your project by creating a separate SQL file for each snapshot table. This file defines the base table or view you're snapshotting, the primary key, and any other necessary configurations.

Here's a simplified example:

```
-- snapshots/customer_snapshot.sql

{{ config(
  materialized='snapshot',
  unique_key='customer_id',
  target_database='analytics',
  target_schema='snapshots',
  strategy='timestamp'
) }}

SELECT
  customer_id,
  name,
  email,
  address,
  current_timestamp() as snapshot_timestamp
FROM
  source.customer;
```

24.What is macros?

Answer: macros refer to reusable blocks of SQL code that can be defined and invoked within dbt models. dbt macros are similar to functions or procedures in other programming languages, allowing you to encapsulate and reuse SQL logic across multiple queries.

Here's how dbt macros work:

1. **Definition:** A macro is defined in a separate file with a `.sql` extension. It contains SQL code that can take parameters, making it flexible and reusable.

```
-- my_macro.sql
{% macro my_macro(parameter1, parameter2) %}
SELECT
  column1,
  column2
FROM
  my_table
WHERE
  condition1 = {{ parameter1 }}
  AND condition2 = {{ parameter2 }}
{% endmacro %}
```

2. **Invocation:** You can then use the macro in your dbt models by referencing it.

```
-- my_model.sql
{{ my_macro(parameter1=1, parameter2='value') }}
```

When you run the dbt project, dbt replaces the macro invocation with the actual SQL code defined in the macro.

3. **Parameters:** Macros can accept parameters, making them dynamic and reusable for different scenarios. In the example above, `parameter1` and `parameter2` are parameters that can be supplied when invoking the macro.

4. **Code Organization:** Macros help in organizing and modularizing your SQL code. They are particularly useful when you have common patterns or calculations that need to be repeated across multiple models.

```
-- my_model.sql
{{ my_macro(parameter1=1, parameter2='value') }}

-- another_model.sql
{{ my_macro(parameter1=2, parameter2='another_value') }}
```

25.what is project structure?

Answer: Aproject structure refers to the organization and layout of files and directories within a dbt project. dbt is a command-line tool that enables data analysts and engineers to transform data in their warehouse more effectively. The project structure in dbt is designed to be modular and organized, allowing users to manage and version control their analytics code easily.

A typical dbt project structure includes the following key components:

1. **Models Directory:**

This is where you store your SQL files containing dbt models. Each model represents a logical transformation or aggregation of your raw data. Models are defined using SQL syntax and are typically organized into subdirectories based on the data source or business logic.

2. **Data Directory:**

The `data` directory is used to store any data files that are required for your dbt transformations. This might include lookup tables, reference data, or any other supplemental data needed for your analytics.

3. **Analysis Directory:**

This directory contains SQL files that are used for ad-hoc querying or exploratory analysis. These files are separate from the main models and are not intended to be part of the core data transformation process.

4. **Tests Directory:**

dbt allows you to write tests to ensure the quality of your data transformations. The `tests` directory is where you store YAML files defining the tests for your models. Tests can include checks on the data types, uniqueness, and other criteria.

5. **Snapshots Directory:**

Snapshots are used for slowly changing dimensions or historical tracking of data changes. The `snapshots` directory is where you store SQL files defining the logic for these snapshots.

6. **Macros Directory:**

Macros in dbt are reusable pieces of SQL code. The `macros` directory is where you store these macros, and they can be included in your models for better modularity and maintainability.

7. **Docs Directory:**

This directory is used for storing documentation for your dbt project. Documentation is crucial for understanding the purpose and logic behind each model and transformation.

8. **dbt_project.yml:**

This YAML file is the configuration file for your dbt project. It includes settings such as the target warehouse, database connection details, and other project-specific configurations.

9. **Profiles.yml:**

This file contains the connection details for your data warehouse. It specifies how to connect to your database, including the type of database, host, username, and password.

10. **Analysis and Custom Folders:**

You may have additional directories for custom scripts, notebooks, or other artifacts related to your analytics workflow.

Having a well-organized project structure makes it easier to collaborate with team members, maintain code, and manage version control. It also ensures that your analytics code is modular, reusable, and easy to understand.

```
my_project/
|-- analysis/
|   |-- my_analysis_file.sql
|-- data/
|   |-- my_model_file.sql
|-- macros/
|   |-- my_macro_file.sql
|-- models/
|   |-- my_model_file.sql
|-- snapshots/
|   |-- my_snapshot_file.sql
|-- tests/
|   |-- my_test_file.sql
|-- dbt_project.yml
```

26. What is data refresh?

Answer: "data refresh" typically refers to the process of updating or reloading data in your data warehouse. Dbt is a command-line tool that enables data analysts and engineers to transform data in their warehouse more effectively. It allows you to write modular SQL queries, called models, that define transformations on your raw data.

Here's a brief overview of the typical workflow involving data refresh in dbt:

1. **Write Models:** Analysts write SQL queries to transform raw data into analysis-ready tables. These queries are defined in dbt models.
2. **Run dbt:** Analysts run dbt to execute the SQL queries and create or update the tables in the data warehouse. This process is often referred to as a dbt run.
3. **Data Refresh:** After the initial run, you may need to refresh your data regularly to keep it up to date. This involves re-running dbt on a schedule or as needed to reflect changes in the source data.
4. **Incremental Models:** To optimize performance, dbt allows you to write incremental models. These models only transform and refresh the data that has changed since the last run, rather than reprocessing the entire dataset. This is particularly useful for large datasets where a full refresh may be time-consuming.
5. **Dependency Management:** Dbt also handles dependency management. If a model depends on another model, dbt ensures that the dependencies are run first, maintaining a proper order of execution.

By using dbt for data refresh, you can streamline and automate the process of transforming raw data into a clean, structured format for analysis. This approach promotes repeatability, maintainability, and collaboration in the data transformation process.

Written by nishad patkar

25 Followers

Follow

More from nishad patkar

nishad patkar

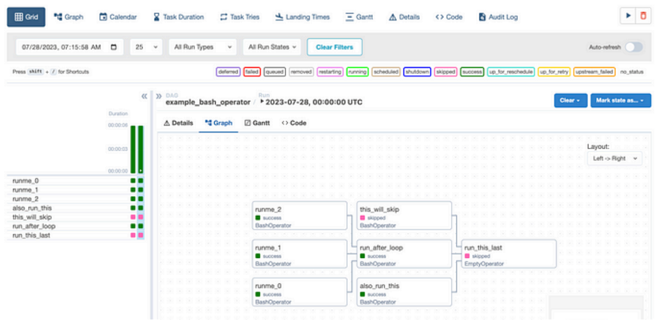
How Cloud Formation Works?

AWS CloudFormation is a service provided by Amazon Web Services (AWS) that allows you to define and provision infrastructure as code. With...

Dec 29, 2023

See all from nishad patkar

Recommended from Medium



Sai Parvathaneni in Towards Dev

Building a Data Modeling Pipeline - dbt, Snowflake, and Airflow

In this article, we'll build a Data Modeling pipeline using dbt, Snowflake, and Airflow. B...

Aug 25 28

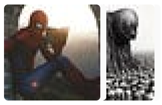
Nouman in Python in Plain English

Automating Data Engineering Pipelines with Apache Airflow

Using Apache Airflow to Automate and Orchestrate Data Engineering Tasks in...

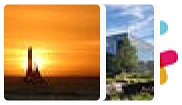
Apr 2 1

Lists



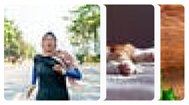
Staff Picks

727 stories · 1275 saves



Stories to Help You Level-Up at Work

19 stories · 780 saves



Self-Improvement 101

20 stories · 2677 saves

Productivity 101

20 stories · 2297 saves

Kamireddy Mahendra in MeanLifeStudies

Hugo Lu

Meta (Hard Level) SQL Interview Question — Solution

Solve SQL Interview Problems asked in MAANG Companies.

Aug 27 10 3

Running dbt-core on Github Actions: a comprehensive guide

Github runners are surprisingly versatile, but peel back the onion and you'll realise...

Apr 18 16

Nnamdi Samuel in Art of Data Engineering

SQL Query Optimization Tips I Regret Not Knowing Earlier

15 Tricks to Level-Up Your Data Game

Aug 27 4

Feruz Urazaliev

Unlocking Databricks: 10 Hidden Features Every Data Engineer...

Databricks is a powerful cloud-based data platform that integrates seamlessly with...

Aug 20 16 2

See more recommendations