1. What is dbt and why is it used?
dbt (data build tool) is a command-line tool and development framework that enables analysts and engineers to transform raw data into clean, tested, and documented datasets in the data warehouse. It's used primarily for **data transformation**, enabling SQL-based modeling with version control, testing, and documentation.

---

2. How does dbt fit into the modern data stack?
dbt sits in the **T (Transform)** layer of the modern ELT pipeline. With tools like Fivetran or Airbyte handling extraction and loading, dbt transforms raw data into business-ready datasets inside the warehouse (like Snowflake, BigQuery, or Redshift).

---

3. How is dbt different from traditional ETL tools?

- dbt only does **T (transform)** and relies on the warehouse for compute.
- Traditional ETL tools are often monolithic and handle E, T, and L together.
- dbt promotes **software engineering best practices** (like modularity, version control, CI/CD).
- It's **declarative** — you declare dependencies and dbt builds the DAG.

---

4. What is the role of SQL in dbt?
All transformations in dbt are written in SQL. dbt layers **Jinja templating** on top of SQL, enabling dynamic logic, macro reuse, and cleaner code. It empowers analysts to work as engineers using a language they already know.

---

5. What are the different materializations in dbt and when do you use each?

- `view`: Default, fast to build, always reflects the latest data. Good for lightweight logic.
- `table`: Materialized as a full table. Use for intermediate or final datasets where performance matters.
- `incremental`: Appends or updates data based on logic. Best for large datasets to reduce cost and runtime.
- `ephemeral`: Not materialized; gets inlined as a CTE. Good for reusable logic or simple transformations.

---

6. What is `ref()` in dbt and how does it work?
`ref('model_name')` is used to reference another dbt model. It tells dbt to include that model as a dependency in the DAG and ensures it's built in the right order.

---

7. What is the difference between `ref()` and `source()`?

- `ref()` is used to reference other dbt models.
- `source()` is used to reference raw tables that come from outside of dbt, defined in `sources` in `schema.yml`.

---

8. How does dbt build the DAG and manage model dependencies?
dbt constructs the DAG by analyzing `ref()` calls across models. It parses dependencies and builds models in the correct order. The DAG also powers dbt's selective runs (`--select`, `--exclude`, etc.).

---

9. What happens if you reference a model that doesn't exist using `ref()`?
dbt will throw a compilation error during parsing, preventing the run. It ensures all `ref()` dependencies are valid before execution.

---

10. What naming conventions do you follow in dbt projects?

- Prefixes like `stg_`, `int_`, `fct_`, `dim_`, `snap_` for clarity.
- Use lowercase with underscores.
- Keep names meaningful, e.g., `stg_orders`, `dim_customer`.

---

11. Can you describe a dbt project you've worked on?
[Adapt this to your experience.] Example: "I built a customer analytics mart using Snowflake and dbt. I ingested data from Stripe and Salesforce, staged them using `stg_` models, built logic in `int_` models, and exposed business-ready tables like `fct_orders`, `dim_customers`, and `fct_subscriptions`."

---

12. How did you structure your models using staging, intermediate, and marts layers?

- **staging**: Cleaned raw data from sources (renamed, casted types).
- **intermediate**: Business logic like joins, calculated fields.
- **marts**: Final models for reporting (fact and dimension tables).

---

13. How many models were in your last project, and how were they organized?
[Customize.] "Roughly 80 models. ~30 staging models, ~25 intermediate, and ~25 in the marts layer. Organized in folders by domain (customers, orders, products, etc.)."

---

14. What are staging models and why are they important?
They clean and standardize raw data, apply naming conventions, and prepare tables for downstream use. They decouple source schema changes from business logic.

---

15. What is an incremental model in dbt and how does it work?
An incremental model updates only new or changed data using logic defined with `is_incremental()` in your SQL. dbt tracks previous runs and only processes new data, improving performance.

---

16. How do you use `is_incremental()` in your logic?

```
sql
CopyEdit
WHERE updated_at > (SELECT MAX(updated_at) FROM {{ this }})
```

Wrap this in an `if is_incremental()` block to apply the filter only during incremental runs.

---

17. How do you track updates using `updated_at` or `inserted_at` columns?
I use `updated_at` to filter rows during incremental loads. For SCDs or snapshots, it helps determine if a row has changed.

---

18. What are common issues with incremental models, and how do you resolve them?

- Missing `unique_key` causes duplicates
- Schema changes in source
- Data drift
  Fixes: Add tests, use `on_schema_change`, validate logic with full-refresh, monitor row counts.

---

19. How do you handle slowly changing dimensions (SCD Type 1 or 2) in dbt?

- Type 1: Use incremental model with overwrite logic.
- Type 2: Use **dbt snapshots** with `check` or `timestamp` strategy to track historical changes.

---

20. What types of tests are available in dbt?

- **Generic tests**: `unique`, `not_null`, `accepted_values`.
- **Custom tests**: You write SQL to define your own test logic.

---

21. What's the difference between generic and custom tests in dbt?

- Generic tests are reusable and declared in YAML.
- Custom tests are raw SQL queries returning rows that fail the test.

---

22. How do you organize `schema.yml` files in large projects?
Each model folder has its own `schema.yml`. This keeps definitions close to models and improves maintainability.

---

23. How do you react to test failures in production?
I set up CI pipelines to catch failures early. For critical models, test failures block deploys. In production, we alert, investigate, and hotfix as needed.

---

24. What are dbt snapshots and when would you use them?
Snapshots are used to track historical changes in slowly changing dimensions (SCD Type 2). They capture row versions over time.

---

25. What's the difference between `check` and `timestamp` snapshot strategies?

- `check`: Compares columns to detect changes.
- `timestamp`: Relies on a timestamp column to detect updates.

---

26. How do you configure a snapshot for SCD Type 2?

```
yml
CopyEdit
strategy: timestamp
updated_at: updated_at
unique_key: id
```

---

27. How do you use Jinja templating in dbt models?
For dynamic logic, loops, conditions, macros, and DRY code. Example: wrapping filters in `is_incremental()`, calling macros, building dynamic SQL.

---

28. Have you created any custom macros in dbt?
Yes — I've built macros for standard date filters, logging, and reusable CTEs.

---

29. How do you pass arguments to macros?
Macros accept keyword arguments:

```
sql
CopyEdit
{% macro filter_by_date(column, days) %}
WHERE {{ column }} >= current_date - interval '{{ days }} day'
{% endmacro %}
```

---

30. What's the difference between a macro and a model in dbt?

- Macro: A reusable Jinja function, not materialized.
- Model: A SQL file materialized as a view, table, etc.

---

31. How do you manage multiple environments like dev, staging, and prod in dbt?
Using `profiles.yml` with different targets (dev/staging/prod). I also use environment variables and separate schemas for each environment.

---

32. How do you manage your `profiles.yml` for different targets?
Profiles define connections and credentials per target. I use `target` blocks and environment variables to dynamically switch between environments.

---

33. Do you use dbt Cloud, dbt CLI, or both? Why?
I've used both.

- **dbt Cloud** for scheduling, logging, and team collaboration.
- **CLI** for local dev, custom CI/CD, and flexibility in pipelines.

---

34. How do you deploy dbt models with orchestration tools like Airflow or Prefect?
I use CLI commands in DAGs. For example, in Airflow:

```
bash
CopyEdit
dbt run --select tag:nightly
```

I monitor via logs and use sensors to wait for completion.

---

35. Have you implemented CI/CD for dbt? What tools did you use?
Yes — using GitHub Actions and dbt Cloud jobs. I validate PRs using `dbt compile`, `dbt test`, and `dbt build`.

---

36. How do you troubleshoot a failed dbt run?

- Review logs
- Identify failing model and its dependencies
- Use `--select` to isolate model
- Run `dbt debug` and `dbt compile` to validate

---

37. What is your debugging process when a model fails?
Start with `dbt compile` and check SQL
Isolate the model
Add `limit` to inspect data
Check upstream models and logs
Validate source freshness

---

38. How do you use `--select`, `--exclude`, and tags in dbt run commands?

- `--select model_name` runs specific model
- `--exclude staging` skips staging
- `--select tag:nightly` runs tagged models

---

39. How do you improve performance for large dbt models?

- Use incremental models
- Optimize SQL (avoid cross joins, CTE bloat)
- Partition large tables
- Leverage warehouse features like clustering

---

40. When would you prefer `table` over `incremental`, even if data grows?

- When source data changes frequently and full reloads are safer
- For small to mid-sized datasets
- When logic is too complex to make incremental

---

41. How do you optimize warehouse costs in Snowflake or BigQuery using dbt?

- Avoid full-refresh unless needed
- Use incremental builds
- Materialize heavy models as `table`, lightweight ones as `view`
- Monitor model runtime and warehouse usage

---

42. How do you document models and columns in dbt?
In `schema.yml` files using `description` fields. I also use `meta` for additional metadata like owners, sensitivity, etc.

---

43. What does `dbt docs generate` do?
It compiles documentation for your project, including models, columns, tests, and DAG. You can view it using `dbt docs serve`.

---

44. How do you maintain governance and consistency in a multi-user dbt project?

- Enforce naming conventions
- Use pre-commit hooks
- Code reviews
- CI checks
- Centralized documentation and testing standards

---

45. What is the use of the `meta` field in `schema.yml`?
It stores arbitrary metadata (e.g., column sensitivity, owners, tags), which can be used in custom macros or governance tools.

---

46. What are dbt artifacts like `manifest.json` and `run_results.json`, and how do you use them?

- `manifest.json`: DAG, metadata, dependencies
- `run_results.json`: Status of each model/test
  Used for CI/CD, lineage, and integrations with tools like Great Expectations or dashboards.

---

47. Have you used the `dbt_utils` package? What are your favorite macros from it?
Yes — favorites include:

- `surrogate_key()`
- `get_column_values()`
- `date_spine()`
- `star()`

---

48. Have you used other dbt packages like `dbt_expectations` or `audit_helper`?
Yes — `dbt_expectations` for expressive tests, `audit_helper` for data validation and completeness checks.

---

49. What are seeds in dbt and when should you use them?
Seeds are CSV files version-controlled and loaded as tables. Useful for static mappings, test data, or lookup tables.

---

50. How do you manage version control in a team using dbt?
Using Git (GitHub/GitLab). Each dev works on feature branches. We use pull requests with mandatory code reviews and CI checks.

---

51. How do you conduct code reviews for dbt model changes?
Review SQL quality, naming, tests, documentation, and performance. Ensure adherence to team standards and logic correctness.

---

52. How do you onboard new team members to a dbt project?
Provide onboarding docs, walkthroughs of model layers, dev setup guides, and pair programming sessions. I also explain testing, CI/CD, and deployment flows.

---

53. Can you give an example where dbt simplified a pipeline or improved reliability?
Yes — we replaced a complex ETL workflow with dbt models. The modular structure and built-in testing made it easier to debug, maintain, and trust. Time to deliver insights dropped significantly.