☑ 2. dbt — 50 preguntas

1. ¿Qué es dbt y cuál es su propósito principal dentro de un pipeline de datos?

dbt (data build tool) es una herramienta que permite **transformar datos en un data warehouse usando SQL de manera modular, reproducible y versionada**.

- **Propósito:** Implementar la transformación de datos (T del ELT) de forma confiable y mantenible, automatizando tests y documentación.
- **Ejemplo:** Transformar tablas crudas de ventas (`raw_sales`) en una tabla agregada lista para análisis (`fact_sales`).

---

2. Explica la diferencia entre modelos ephemeral, table y view

- **ephemeral:** No crea objeto físico en el DWH; se inyecta el SQL en el query que lo llama.
- **table:** Crea tabla física en el warehouse; datos persistentes.
- **view:** Crea una vista; no almacena datos, se calcula al consultar.
- **Ejemplo:**

```
-- ephemeral
{{ config(materialized='ephemeral') }}
SELECT * FROM raw_sales
```

---

3. ¿Cómo funciona el mecanismo de ref() en dbt?

`ref('modelo')` genera una **dependencia entre modelos**, resolviendo automáticamente el orden de ejecución y el schema del modelo.

- **Ejemplo:**

```
SELECT * FROM {{ ref('dim_customer') }}
```

Esto asegura que `dim_customer` se construya antes del modelo actual.

---

4. ¿Qué es el manifest.json y para qué se utiliza?

Archivo generado por dbt que contiene **metadata completa del proyecto**: modelos, tests, dependencias y relaciones.

- **Uso:** Debug, herramientas externas, documentación y análisis de lineage.

---

5. ¿Cómo implementar tests personalizados en dbt?

1. Crear un archivo `.sql` en `tests/`.
2. Escribir la lógica que devuelve filas que **fallan la condición**.
3. Invocarlo en YAML con `test:`.

- **Ejemplo:** Test que valida que el email tenga formato correcto.

---

6. Describe el flujo de un run de dbt desde inicio a fin

1. Cargar configuraciones del proyecto.
2. Parsear modelos y dependencias (`ref`).
3. Compilar SQL en el warehouse.
4. Ejecutar modelos según orden de dependencias.
5. Correr tests si se indica.
6. Generar artifacts (manifest.json, run_results.json).

---

7. Tipos de tests nativos

- **unique:** No se repiten valores en columna.
- **not_null:** Columna no tiene valores nulos.
- **accepted_values:** Valores deben pertenecer a un conjunto definido.
- **relationships:** Verifica integridad referencial entre tablas.

---

8. ¿Cómo se manejan variables en dbt (vars)?

- Se definen en `dbt_project.yml` o al correr dbt con `--vars`.
- Acceso dentro de SQL con `{{ var('nombre_var', 'default') }}`.
- **Ejemplo:** Cambiar dinámicamente la fecha de corte en un modelo.

---

9. ¿Qué son los sources en dbt y para qué sirven?

- Referencias a **datos originales del warehouse** (raw).
- Permiten tests y lineage sobre datos de entrada.

- **Ejemplo:** `source('raw', 'sales')` para apuntar a tabla cruda de ventas.

---

10. Diferencia entre sources y seeds

- **source:** Tabla externa ya existente en el warehouse.
- **seed:** Archivo CSV versionado dentro del proyecto que dbt carga como tabla.
- **Ejemplo:** Seed para códigos de país (`countries.csv`).

---

11. ¿Cómo documentar modelos con YAML?

- Crear archivo `.yml` en `models/` con `description:` y columnas.

```
models:
  - name: dim_customer
    description: "Tabla de clientes"
    columns:
      - name: customer_id
        description: "ID único del cliente"
```

---

12. ¿Cómo se genera la documentación HTML con dbt docs?

- Comando:

```
dbt docs generate
dbt docs serve
```

- Permite explorar lineage, modelos, tests y sources en interfaz web.

---

13. ¿Qué es el state comparison de dbt y cuándo se usa?

- Comparación entre **estado anterior y actual** del proyecto.
- Útil para **tests selectivos** o despliegues incrementales.
- **Ejemplo:** Solo ejecutar tests de modelos que cambiaron desde la última ejecución.

---

14. Explica snapshotting en dbt y cuál es su objetivo

- Snapshots almacenan **historial de cambios en tablas fuente** (SCD2).
- Objetivo: **rastrear cambios históricos sin perder datos anteriores**.

- **Ejemplo:** Registro de cambios de dirección de clientes.

---

15. Diferencias entre timestamp y check strategy en snapshots

- **timestamp:** Basado en columna de fecha/hora de actualización.
- **check:** Detecta cambios comparando columnas específicas.

---

16. ¿Cómo funciona incremental en dbt?

- Actualiza solo **nuevos registros** en lugar de reconstruir toda la tabla.
- Requiere clave única (`unique_key`) y lógica de merge.
- **Ejemplo:** Fact table de ventas que solo agrega las ventas del día.

---

17. ¿Qué pasa si cambias la clave incremental de un modelo?

- dbt reconstruye toda la tabla incremental porque la nueva clave rompe la lógica de merge.
- Posible pérdida de historial si no se maneja correctamente.

---

18. ¿Cómo depurar un modelo en dbt?

- Usar `dbt compile` para revisar SQL generado.
- Ejecutar modelo individual con `dbt run -m modelo`.
- Revisar logs en `logs/dbt.log`.

---

19. ¿Qué son los hooks (pre-hook y post-hook)?

- **pre-hook:** Ejecuta SQL antes de construir modelo.
- **post-hook:** Ejecuta SQL después.
- **Ejemplo:** Registrar auditoría de creación de tabla, o crear índices.

---

20. Explica el concepto de packages en dbt

- Paquetes de modelos, macros o tests reutilizables.
- Permiten compartir código entre proyectos.
- **Ejemplo:** `dbt_utils` ofrece tests y macros listas para usar.

---

21. ¿Cómo crear un package privado?

1. Crear proyecto dbt y versionarlo en Git.
2. Publicar URL en `packages.yml` del proyecto consumidor.
3. Instalar con `dbt deps`.

---

22. ¿Qué es dbt build y cómo difiere de dbt run + dbt test?

- `dbt build` combina **run, test, snapshot y seed** en un solo comando.
- Más conveniente para pipelines completos.

---

23. ¿Cómo funciona exposures?

- Define **consumidores de modelos dbt**: dashboards, ML models, apps.
- Permite lineage hacia usuarios finales.
- **Ejemplo:** Dashboard de ventas que depende de `fact_sales`.

---

24. ¿Cómo integras dbt con herramientas de orquestación?

- Usar comandos `dbt run`, `dbt test` dentro de DAGs o workflows.
- Ejemplo: Airflow operator `BashOperator` con `dbt run --models fact_sales`.

---

25. Ventajas del versionamiento del código SQL usando dbt

- Control de cambios, rollback y colaboración en equipo.
- Integración con CI/CD para despliegues automatizados.

---

26. Explica el lineage graph

- Visualiza **dependencias entre modelos y fuentes**.
- Útil para depurar, planificar cambios y documentar.

---

27. ¿Cómo manejar múltiples ambientes (dev, test, prod) en dbt?

- Usar `profiles.yml` con credenciales por ambiente.
- Cambiar variables o schemas según ambiente.

---

28. ¿Qué es un project evaluator en dbt Cloud?

- Herramienta que **valida configuraciones, dependencias y errores** antes de ejecutar runs.

---

29. Diferencias entre dbt Core y dbt Cloud

Característica	dbt Core	dbt Cloud
Interfaz	CLI	Web UI + Scheduler
Orquestación	Externa	Interna
Costo	Gratis	Pago

---

30. ¿Cómo funcionan los macros en dbt y cuándo usarlos?

- Bloques de SQL reutilizables con Jinja.
- Útiles para cálculos repetidos, tests o queries dinámicas.
- **Ejemplo:** Macro que devuelve fecha de corte.

---

31. Explica Jinja y su uso dentro de dbt

- Lenguaje de templates que permite **insertar variables, loops, condicionales y funciones**.
- **Ejemplo:**

```
SELECT * FROM {{ ref('dim_customer') }} WHERE country = '{{
var("country") }}'
```

---

32. ¿Cómo crear tests generados por macros?

- Crear macro que genere SQL de test dinámicamente.
- Llamarlo en YAML usando `test:`.
- Permite parametrizar tests según columnas o reglas.

---

33. ¿Cómo implementar un sistema de roles y permisos con dbt?

- Configurar `grants` en modelos.
- Asignar select, insert, update a roles específicos.
- **Ejemplo:** Analistas solo pueden leer dashboards, ETL engineers pueden escribir.

---

34. ¿Qué problemas típicos resuelve un persist_docs?

- Documenta **columnas y modelos** en el warehouse.
- Útil para auditoría y BI.
- Evita duplicar descripciones entre dbt y herramientas externas.

---

35. ¿Qué son los artifacts?

- Archivos generados por dbt durante ejecución: `manifest.json`, `run_results.json`, `catalog.json`.
- Contienen metadata, resultados de tests y lineage.

---

36. Errores comunes al correr dbt y cómo solucionarlos

- Error de dependencia: revisar `ref()`.
- Falla incremental: revisar `unique_key`.
- Problemas de permisos: ajustar roles en warehouse.

---

37. ¿Cómo optimizar modelos muy costosos?

- Usar incremental, ephemeral y tablas intermedias.
- Optimizar joins y filtros.
- Materializar como table si se usa repetidamente.

---

38. ¿Qué es un semantic layer y qué soporte ofrece dbt?

- Capa que **traduce datos técnicos a términos de negocio**.
- dbt permite documentar modelos, exponer descriptions y lineage a BI.

---

39. ¿Cómo manejar cambios en fuentes que rompen modelos dependientes?

- Actualizar sources.yml y tests.
- Revisar lineage y modelos afectados.
- Aplicar incremental o snapshot según corresponda.

---

40. ¿Qué es el run-operation?

- Comando para ejecutar **macros arbitrarias fuera de modelos**.
- Útil para tareas de mantenimiento o administración.

---

41. ¿Qué es test freshness?

- Valida **que los datos estén actualizados** según un límite de tiempo.
- **Ejemplo:** Asegurar que la tabla de ventas tenga datos del día.

---

42. ¿Cómo funciona dbt seed?

- Carga **archivos CSV versionados** en tablas del warehouse.
- Útil para datos estáticos o lookup tables.

---

43. Estrategia para nombrar modelos

- Prefijos según tipo: `dim_`, `fact_`, `stg_`.
- Consistencia y claridad para facilitar lineage y mantenimiento.

---

44. ¿Qué es un exposure downstream?

- Indica **consumidor final de un modelo dbt**, como dashboards o ML pipelines.
- Permite trazar impacto de cambios en upstream models.

---

45. ¿Cómo gestionar archivos de configuración grandes?

- Dividir en múltiples YAML por carpetas o módulos.
- Usar variables y defaults para reducir repetición.

---

46. ¿Qué son los generic tests?

- Tests parametrizados aplicables a **cualquier modelo o columna**.
- Ejemplo: `unique`, `not_null`, `relationships`.

---

47. ¿Qué es disable model y para qué sirve?

- Deshabilita un modelo temporalmente en ejecución (`enabled: false`).
- Útil para pruebas o cuando un modelo está en mantenimiento.

---

48. ¿Cómo manejar errores cuando un modelo incremental pierde su partición base?

- Reconstruir tabla completa o actualizar la lógica de merge.
- Ajustar `unique_key` y filtros de incremental.

---

49. ¿Cómo implementar SCD2 con dbt?

- Usar **snapshots** con estrategia timestamp o check.
- Registrar cambios históricos con columnas `valid_from`, `valid_to` y `current_flag`.

---

50. ¿Cómo medir el performance de un pipeline de dbt?

- Revisar `run_results.json` para tiempos por modelo.
- Monitorizar warehouse (CPU, memoria, I/O).
- Optimizar modelos largos usando materialización adecuada y incremental.