Dateneo®

Pruebas de datos en dbt: Asegurando la integridad de tus modelos

Las pruebas de datos son afirmaciones que haces sobre tus modelos y otros recursos en tu proyecto dbt (como fuentes, semillas y snapshots). Cuando ejecutas `dbt test`, dbt te informará si cada prueba en tu proyecto pasa o falla.

Puedes usar pruebas de datos para mejorar la integridad del SQL en cada modelo haciendo afirmaciones sobre los resultados generados.

Fundamentos de las pruebas de datos

¿Qué son las pruebas de datos?

Las pruebas de datos son consultas SQL que buscan registros "fallidos" que refutan tu afirmación. Si la prueba devuelve cero filas, pasa y tu afirmación ha sido validada.

Pruebas integradas

dbt incluye cuatro pruebas genéricas: **unique**, **not_null**, **accepted_values** y **relationships**.

Beneficios

Las pruebas de datos confirman que tus salidas y entradas son las esperadas y ayudan a prevenir regresiones cuando tu código cambia.

Las pruebas de datos devuelven un conjunto de registros fallidos. Las pruebas de datos genéricas (también conocidas como pruebas de esquema) se definen usando bloques de prueba.

Singulific Test

Validation Table			
Data	Table	Fabre	Test
☐	Erste		☑
☐	Essole		☑
☐	Erste		☑
☐	Vasian		☑

Generic Test

Validation Check			
Dext	Table	Dam	Test
☐	Validation		☑
☐	Validation		☑
☐	Fection		☑
☐	Validation		☑

Dos tipos de pruebas

Pruebas Singulares

- La forma más simple de definir una prueba
- Archivos SQL en el directorio de pruebas
- Consultas específicas para un solo propósito
- Se ejecutan automáticamente con `dbt test`

Ejemplo de archivo: `tests/assert_total_payment_amount_is_positive.sql`

Pruebas Genéricas

- Reutilizables en múltiples modelos
- Definidas en bloques `test`
- Aceptan parámetros (modelo, columna)
- Se configuran en archivos YAML

Deberían constituir la mayor parte de tu suite de pruebas dbt.

Pruebas genéricas integradas

unique

Verifica que una columna contenga valores únicos.

```
- unique
```

not_null

Verifica que una columna no contenga valores nulos.

```
- not_null
```

accepted_values

Verifica que los valores estén en una lista específica.

```
- accepted_values: values: ['placed', 'shipped', 'completed', 'returned']
```

relationships

Verifica la integridad referencial entre tablas.

```
- relationships: to: ref('customers') field: id
```

Detrás de escena, dbt construye una consulta SELECT para cada prueba de datos, utilizando la consulta parametrizada del bloque de prueba genérica. Estas consultas devuelven las filas donde tu afirmación no es verdadera.

Implementación

Ejemplo de configuración en YAML

```yaml
version: 2

models:
  - name: orders
    columns:
      - name: order_id
        data_tests:
          - unique
          - not_null
      - name: status
        data_tests:
          - accepted_values:
              values: ['placed', 'shipped', 'completed', 'returned']
      - name: customer_id
        data_tests:
          - relationships:
              to: ref('customers')
              field: id
```

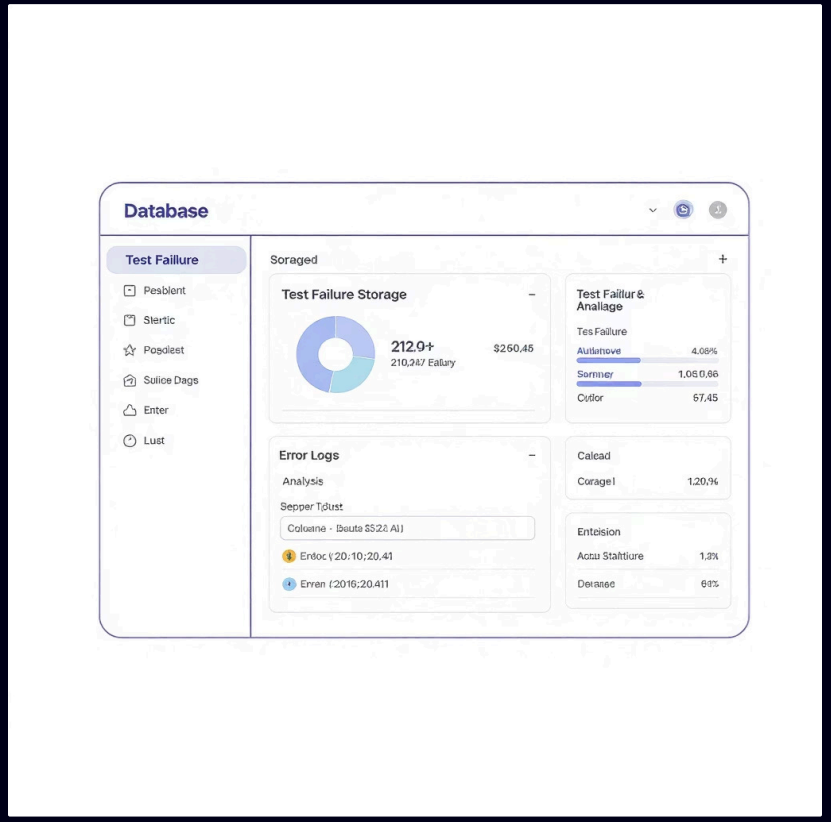
Para ejecutar las pruebas, simplemente usa el comando: $ dbt test

Características avanzadas

Almacenamiento de fallos

Puedes almacenar los resultados de las pruebas fallidas en la base de datos para una depuración más rápida:

- Usa la bandera `--store-failures`
- Configura `store_failures: true`
- Los resultados se guardan en un esquema `dbt_test__audit`



Pruebas adicionales

Puedes extender las pruebas de datos con:

- Pruebas genéricas personalizadas
- Paquetes como dbt-utils y dbt-expectations

La nueva sintaxis utiliza `data_tests:` en lugar de `tests:`, aunque ambas son compatibles por ahora.