# 14. Building with Spellbook — How to Contribute Reusable Models to the Community

In the fast-growing world of onchain analytics, building once and reusing often is the key to scaling insights. That's where **Spellbook** comes in.

In this article, we'll explore how Dune's Spellbook enables analysts to turn queries into community-maintained models—so everyone can benefit from shared logic, clean data, and modular SQL.

---

## 🪄 What Is Spellbook?

Spellbook is the **open-source data modeling layer** behind DuneSQL.

Think of it as a **GitHub for reusable queries**, built using dbt, the industry standard in data transformation. With Spellbook, you can:

- Standardize messy onchain data into clean views
- Build abstractions across protocols, contracts, and versions
- Make your SQL easier to write, maintain, and scale
- Help the entire Dune community by contributing shared models

---

## 📚 Why Use Spellbook?

Without Spellbook:

- Your queries are long, repetitive, and hard to debug.
- Every analyst writes their own logic for the same events.
- Protocol upgrades break older dashboards unless manually updated.

With Spellbook:

- Common logic (like Uniswap swaps or NFT mints) lives in one place.
- Models are version-controlled, tested, and composable.
- Updates to source contracts only require a single model fix.

---

## 🔍 A Real Example: Space ID on BNB Chain

Let's say you're tracking domain name registrations from the Space ID project. But Space ID has upgraded their contracts multiple times—V3 to V9.

Rather than querying seven separate event tables and manually unioning them in every dashboard, you can build a **Spell**:

```
SELECT
  'v3' as version,
  evt_block_time as block_time,
  name,
  owner,
  cost,
  tx_hash
FROM {{ source('spaceid_bnb',
'BNBRegistrarControllerV3_evt_NameRegistered') }}

UNION ALL

-- repeat for v4 to v9
```

Now, **any analyst** can query from `spellbook.spaceid_bnb_registrations` and always get the latest data—regardless of contract changes.

---

# 🛠️ How to Build a Spell

## 1. Set Up Your Environment

Follow the guide here: Set Up Spellbook dbt Locally

You'll need:

- GitHub account
- Python + pipenv
- dbt installed
- The forked `spellbook` repo cloned locally

## 2. Create Your Folder Structure

```bash
models/
  spaceid/
    bnb/
      spaceid_bnb_sources.yml
      spaceid_bnb_schema.yml
      spaceid_bnb_registrations.sql
```

## 3. Define Your Sources

Inside `spaceid_bnb_sources.yml`, declare which raw tables your spell pulls from:

```yaml
version: 2

sources:
  - name: spaceid_bnb
    tables:
      - name: BNBRegistrarControllerV3_evt_NameRegistered
        loaded_at_field: evt_block_time
      - name: BNBRegistrarControllerV4_evt_NameRegistered
        loaded_at_field: evt_block_time
```

## 4. Write Your Spell (SQL)

In `spaceid_bnb_registrations.sql`:

```
{{ config(materialized='view', alias='registrations') }}
```

```
SELECT
  'v3' as version,
  evt_block_time as block_time,
  name,
  owner,
  cost,
  evt_tx_hash as tx_hash
FROM {{ source('spaceid_bnb',
'BNBRegistrarControllerV3_evt_NameRegistered') }}

UNION ALL

-- etc...
```

Use `{{ source(...) }}` to dynamically reference raw tables.

### 5. Add Schema and Metadata

In `spaceid_bnb_schema.yml`:

```
yaml

models:
  - name: spaceid_bnb_registrations
    description: "Combined registrations from V3–V9 Space ID contracts"
    columns:
      - name: name
        description: "Domain name registered"
        tests:
          - unique
```

Add contributors, tags, and other metadata.

### 6. Compile and Test

Use dbt to make sure everything works:

```bash

pipenv shell
dbt compile
```

Test locally in Dune by pasting the compiled SQL into the DuneSQL editor.

---

## ✅ Contributing to the Community

Once your model works:

1. Commit your code in a new Git branch
2. Push it to your GitHub repo
3. Create a Pull Request (PR) to the main Spellbook repo
4. Follow the checklist and wait for review
5. Once merged, your spell is live!

Now every analyst on Dune can benefit from your work.

---

## 📐 Best Practices

- Keep models **narrow and reusable**
- Add tests (e.g., uniqueness of key fields)
- Document thoroughly (especially with evolving contracts)
- Use incremental materialization for large datasets

- Use `dbt_utils` where possible (e.g., for date handling)

---

## 🔮 Why It Matters

Spellbook is more than a tool—it's a shared analytics layer for decentralized data.

By contributing models, you:

- Help protocols build better metrics
- Help the next generation of analysts skip redundant work
- Shape the standards for how we measure onchain activity

---

## 🧙‍♀️ Want to See Real Spells?

Explore hundreds of models already live in the Spellbook repo:

👉 https://github.com/duneanalytics/spellbook

You'll find models for:

- Uniswap v2/v3 swaps
- OpenSea mints and trades
- ERC4337 user operations
- NFT projects
- DAO proposals
- Lido staking and more

---

## 🧭 Up Next

Now that you've seen how to create reusable models, we'll turn our attention to the **Dune API**—where you'll learn how to bring onchain data into full applications with real-time query execution and alerting.

**Next: 15. How to Build an Onchain App Using the Dune API**