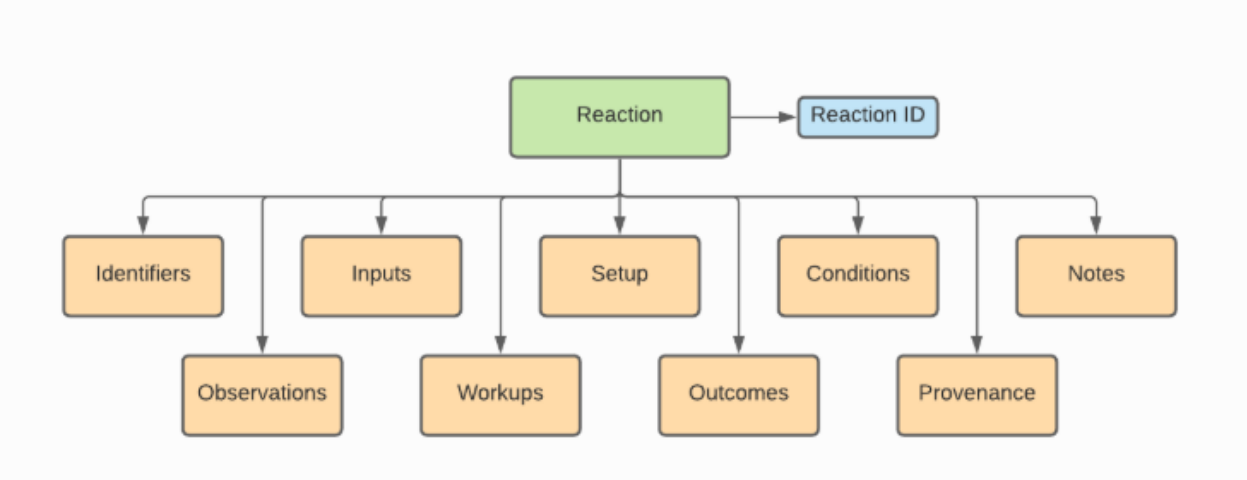
ORD (Open Reaction Database)

What is ORD?

The Open Reaction Database is an open-access schema and infrastructure for structuring and sharing organic chemical reaction data. It is a way to record everything about a chemical reaction, not just what went in and what came out, but all the procedural details: temperatures, stirring rates, vessel types, reaction times, analytical data, and more.

The schema is divided into nine main sections: reaction identifiers, inputs, setup, conditions, notes, observations, workups, outcomes (products and analytics), and provenance.



Why does such a database exist?

Chemistry has a lot of reaction knowledge, but it is mostly in:

- **papers and PDFs** (text + tables),
- **lab notebooks** ,
- and databases that often store only **reaction "equations"** (reactants ⇒ products) without the full context.

The problem before ORD

Most reaction data in chemistry looked like this:

> "Suzuki coupling was performed using $Ni(cod)_2$ and ligand L in dioxane at 60 °C for 2 h, affording the product in 67% yield."

What's missing / broken

- Conditions buried in free text
- Ligands not standardized (same ligand, 10 names)
- No machine-readable structure
- Impossible to systematically compare experiments
- Impossible to reproduce or learn trends at scale

ORD solves this by:

- Storing reactions as **structured records**
- Explicitly encoding:
  - Reactants (SMILES)
  - Catalysts + ligands
  - Solvents
  - Temperature, time, stirring
  - Measured outcomes (yield, analysis method)

What is the problem with other ways of storing information about reactions?

Two reactions can have the *same* equation but completely different outcomes depending on:

- solvent, base, catalyst, temperature, time, atmosphere, etc.

ORD makes those details **explicit and structured**, so experiments become comparable.

The dataset I chose:

Ni-catalyzed C(sp^2)–C(sp^2) Suzuki–Miyaura cross-coupling dataset

- **ORD ID:**
  `ord_dataset-3b5db90e337942ea886b8f5bc5e3aa72`
- **File :**
  `ord-data/data/3b/ord_dataset-3b5db90e337942ea886b8f5bc5e3aa72.pb.gz`
- **Size:**

  450 reactions
- **Reaction class:**

  Ni-catalyzed Suzuki–Miyaura cross-coupling

What are cross coupling reactions?

Cross-coupling is a class of reactions where you **join two carbon fragments together** (or carbon + another element) using a **metal catalyst**.

The basic idea

You start with two pieces:

1. **Organic halide**

- an organic molecule with a leaving group like **Cl, Br, I** attached to carbon
- written as: **R–X** (X = halogen)

1. **Organometallic partner**

- another organic fragment attached to something like **B, Sn, Zn, Mg**
- written as: **R′–M**

A metal catalyst (Pd or Ni) helps swap partners so the two carbon groups connect:

$$\mathrm{R{-}X + R\prime{-}M \xrightarrow[\text{base}]{\text{Ni or Pd}} R{-}R\prime}$$

Cross-coupling is one of the most useful ways to build **C–C bonds**, especially in:

- pharmaceuticals
- agrochemicals
- organic electronics (OLEDs, polymers)

What was installed?

- Python package that defines the **ORD data model**
- Uses **Protocol Buffers (protobuf)**
- Lets you load, inspect, and traverse reactions properly

Why it's needed

- ORD data is **not CSV or JSON**
- It's structured, nested, typed scientific data
- Without this, you cannot interpret reactions, inputs, outcomes, yields, catalysts, etc.

```
pip install ord-schema
```

This is the **essential library**.

What it is

- Python package that defines the **ORD data model**
- Uses **Protocol Buffers (protobuf)**
- Lets you load, inspect, and traverse reactions properly

Why it's needed

- ORD data is **not CSV or JSON**
- It's structured, nested, typed scientific data
- Without this, you cannot interpret reactions, inputs, outcomes, yields, catalysts, etc.

```
from ord_schema.proto import dataset_pb2
```

This imports the **Protocol Buffer schema definitions** which is the "dictionary" that tells Python how to read ORD files.

Loading Data

```
dataset = dataset_pb2.Dataset()  # Create empty dataset object
dataset.ParseFromString(f.read())  # Fill it with data from the .pb.gz file
```

`ord_schema` converts the binary Protocol Buffer file into Python objects you can use.

The tricky part: Enums

Protocol Buffers store enums (like "REACTANT", "CATALYST") as integers. So we need:

```
def get_enum_name(proto_obj, field_name):
    # Converts integer → string name
    # Example: 1 → "REACTANT"
```