External Type Mapping & Selective Override Feature for Ora2Pg

Feature Documentation – BR 1.3.1 / BR 1.3.2

---

1. Background

Ora2Pg handles schema migration from Oracle to PostgreSQL. A key area during migration is Oracle-to-PostgreSQL type translation. Earlier, Ora2Pg allowed only two mechanisms:

- Global type mappings using DATA_TYPE
- Column-level overrides using MODIFY_TYPE

Both had to be defined directly inside ora2pg.conf as inline strings, such as:

DATA_TYPE NUMBER(*\,0):bigint, CLOB:text

MODIFY_TYPE USERS:IS_ACTIVE:boolean

As the number of mappings increased, this became harder to maintain. Inline strings were not structured, were difficult to review, and didn't version-control well.

I implemented two enhancements to address these issues.

---

2. Problem Statement

Before this update, Ora2Pg had several limitations:

- No support for external mapping files (JSON/YAML)
- All mappings had to be embedded in the main config file
- Difficult to manage or audit large mapping sets
- No flexible per-object or per-type override mechanism beyond MODIFY_TYPE
- No direct visibility into which mapping was applied during export

These limitations created maintainability and consistency issues during larger migrations.

---

3. New Feature: External Type Mapping (JSON-Based)

A new configuration directive was introduced:

EXTERNAL_TYPE_MAP = /path/to/custom_types.json

Ora2Pg can now load type mappings from an external JSON file.

The JSON file supports two sections:

Purpose	JSON Key	Format
Global Oracle → PostgreSQL type mappings	DATA_TYPE	"OracleType": "PgType"
Table/column-specific overrides	MODIFY_TYPE	"TABLE": { "COLUMN": "PgType" }

Benefits:

- Clean and readable structure
- Easy version control
- Ability to maintain mapping sets per environment
- No need to modify the main config for mapping changes
- Fully backward compatible

---

4. New Feature: Selective Mapping Mode (Per Object / Per Type)

In BR 1.3.2, I added support for selective mapping rules that apply at:

- Table level
- Column level
- Oracle type level

This is handled through the "MODIFY_TYPE" block in the JSON file.

Example:

```
"MODIFY_TYPE": {
   "ORDERS": {
```

```
    "STATUS": "varchar(20)"
  },
  "PAYMENTS": {
    "AMOUNT": "numeric(12,2)"
  }
}
```

This allows targeted overrides without affecting unrelated tables or columns.

Verification Support

To help with debugging and audits, mapping decisions are logged clearly:

DEBUG: Applying external DATA_TYPE mapping: NUMBER => bigint

DEBUG: Applying external MODIFY_TYPE override: USERS.IS_ACTIVE => boolean

This makes it easy to verify how the final PostgreSQL type was chosen.

---

5. Internal Workflow

Internal processing happens in the following order:

1. Ora2Pg loads its standard configuration.
2. If EXTERNAL_TYPE_MAP is defined:
    - Validates the file
    - Loads the JSON
    - Extracts DATA_TYPE and MODIFY_TYPE sections
3. Merges JSON rules with existing internal mappings
4. Applies the selective override logic
5. Logs each mapping application
6. The SQL export uses the final resolved types

The export process itself is unchanged.

---

6. JSON File Example

```
{
  "DATA_TYPE": {
    "NUMBER(*,0)": "bigint",
    "CLOB": "text",
    "VARCHAR2": "varchar"
  },
  "MODIFY_TYPE": {
    "USERS": {
      "IS_ACTIVE": "boolean",
      "CREATED_AT": "timestamp with time zone"
    },
    "TASKS": {
      "PRIORITY_LEVEL": "smallint",
      "DUE_DATE": "date"
    }
  }
}
```

Both sections are optional.

---

7. Enabling the Feature

Add this line inside ora2pg.conf:

EXTERNAL_TYPE_MAP = /path/to/custom_types.json

No other changes are needed.

---

8. Mapping Priority (Final Resolution Order)

The resolver checks mappings in this sequence:

1. Built-in default mappings
2. DATA_TYPE from config
3. DATA_TYPE from JSON
4. MODIFY_TYPE from config
5. MODIFY_TYPE from JSON (highest priority)

Column-level overrides always win over everything else.

---

9. Benefits Delivered

- Mapping rules are easier to maintain
- Configuration file stays clean
- Better suited for enterprise-level migrations
- Full control over per-table and per-column mappings
- Easy to verify mapping decisions through debug logs
- Full backward compatibility

---

10. Code Diff

1.)

```
# Added these lines near other use statements (around line 28)
+use JSON qw(decode_json);
+use File::Slurp qw(read_file);
```

2.)

```
# Added this in the configuration parsing section (around line 14969)
+elsif ($var eq 'EXTERNAL_TYPE_MAP') {
```

```
+    $AConfig{EXTERNAL_TYPE_MAP} = $val;
+}
```

3.)

```
# Added this new method (around line 1600)
+sub load_external_type_mappings {
+    my ($self) = @_;
+
+    return unless $self->{external_type_map};
+
+    $self->logit("Loading external type mappings from $self->{external_type_map}\n", 1);
+
+    # Check if file exists and is readable
+    unless (-r $self->{external_type_map}) {
+        $self->logit("WARNING: External type map file not found or not readable: $self->{external_type_map}\n", 0);
+        return;
+    }
+
+    # Read and parse JSON
+    my $json_text;
+    eval {
+        $json_text = read_file($self->{external_type_map});
+        my $type_maps = decode_json($json_text);
+
+        # Process DATA_TYPE mappings
```

```
+        if (exists $type_maps->{DATA_TYPE} && ref($type_maps->{DATA_TYPE}) eq 'HASH') {
+            while (my ($ora_type, $pg_type) = each %{$type_maps->{DATA_TYPE}}) {
+                $self->{data_type}{uc($ora_type)} = lc($pg_type);
+                $self->logit("DEBUG: Applying external DATA_TYPE mapping: $ora_type => $pg_type\n", 2);
+            }
+        }
+
+        # Process MODIFY_TYPE overrides
+        if (exists $type_maps->{MODIFY_TYPE} && ref($type_maps->{MODIFY_TYPE}) eq 'HASH') {
+            while (my ($table, $columns) = each %{$type_maps->{MODIFY_TYPE}}) {
+                next unless ref($columns) eq 'HASH';
+                while (my ($column, $type) = each %$columns) {
+                    $self->{modify_type}{lc($table)}{lc($column)} = lc($type);
+                    $self->logit("DEBUG: Applying external MODIFY_TYPE override: $table.$column => $type\n", 2);
+                }
+            }
+        }
+    };
+
+    if ($@) {
+        $self->logit("ERROR: Failed to parse external type map: $@\n", 0, 1);
+    }
```

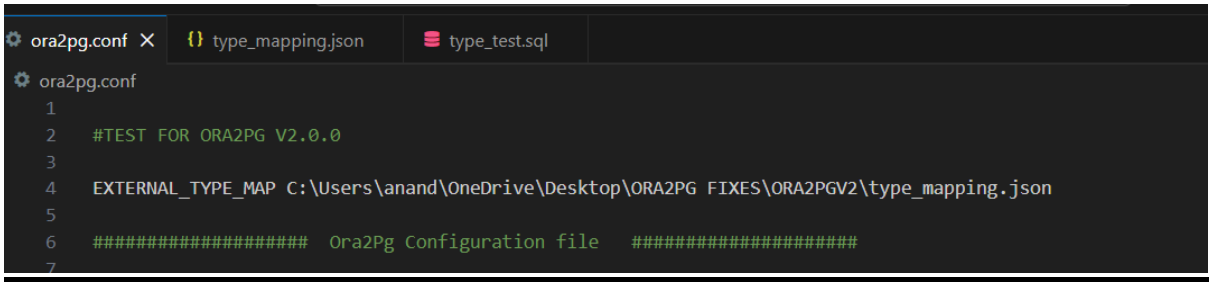
+}

4.)

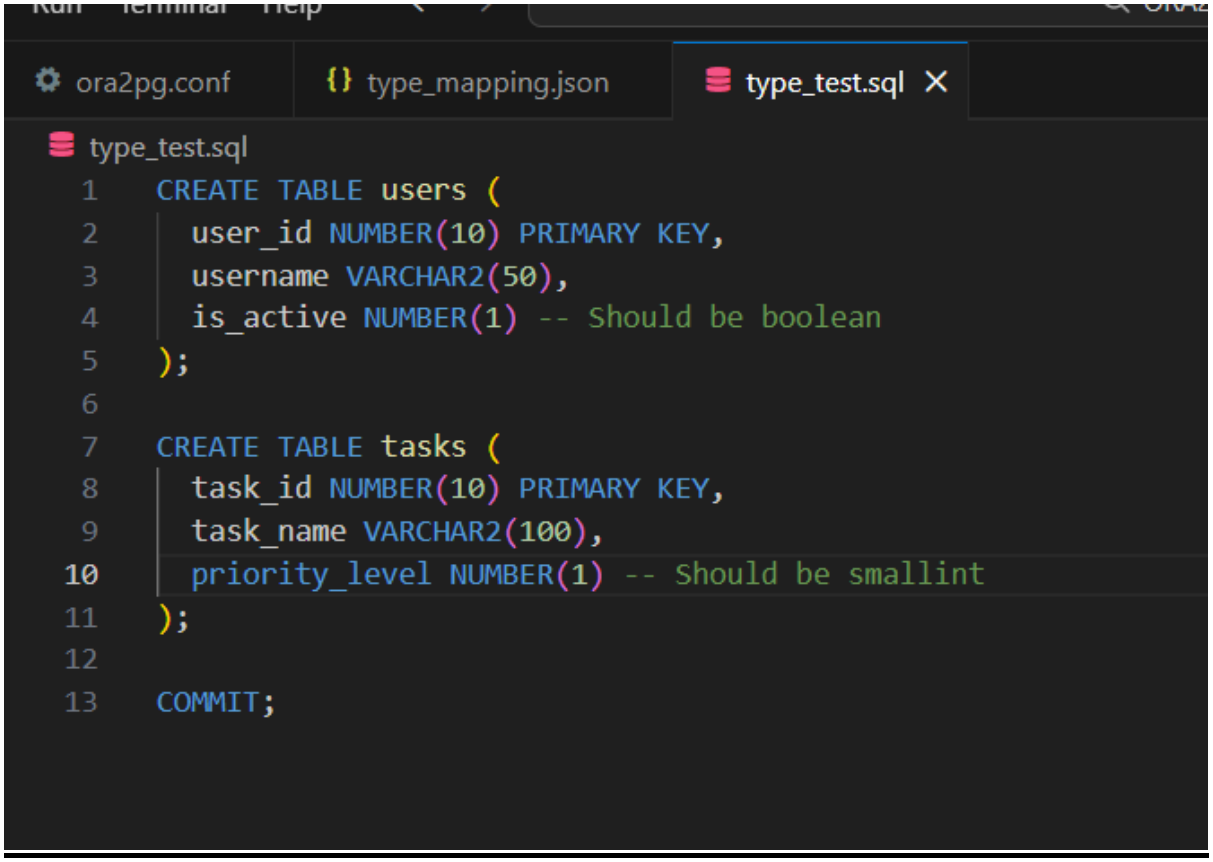
```
# Add this after data_type initialization (around line 1600)
+    # Load external type mappings if configured
+    if ($self->{external_type_map}) {
+        $self->load_external_type_mappings();
+    }
```

SCREENSHOTS:

New derivative for external_type_map Jason given in config file.



Test case schema:



JSON FILE FOR DATA TYPE MAPPING

type_mapping.json

```
{
  "DATA_TYPE": {
    "NUMBER(*,0)": "smallint"
  },
  "MODIFY_TYPE":{
    "users":{
      "is_aCtive":"boolean"
    },
    "TASKS":{
      "priority":"smallint"
    }
  }
}
```

ORA2PG CONVERSION IN PROCESS:

```
PS C:\Users\anand\OneDrive\Desktop\ORA2PG FIXES\ORA2PGV2> ora2pg -t TABLE -o output_test2.sql -c ora2pg.conf
[2025-11-14 16:32:53] [=====================>] 2/2 tables (100.0%) end of scanning.
[2025-11-14 16:32:53] [=====================>] 2/2 tables (100.0%) end of table export.
```

type_test.sql | output_test2.sql

output_test2.sql

```sql
-- Generated by Ora2Pg, the Oracle database Schema converter,
-- Copyright 2000-2025 Gilles DAROLD. All rights reserved.
-- DATASOURCE: dbi:Oracle:host=localhost;port=1522;sid=XE

SET client_encoding TO 'UTF8';

\set ON_ERROR_STOP ON



CREATE TABLE users (
    user_id smallint NOT NULL,
    username varchar(50),
    is_active boolean
) ;

CREATE TABLE tasks (
    task_id smallint NOT NULL,
    task_name varchar(100),
    priority_level smallint
) ;

```

FINAL OUTPUT: