EHS Core Inform Response Data — BI Reporting Guide

What Changed and Why

Ideagen EHS Core has improved the performance of Inform forms. As part of this improvement, the way form responses are stored internally has changed.

Before this update, every answer to every question was stored as an individual row in the `cs_inform_response` table. For a form with 50 questions, saving it created 50 rows.

After this update, all answers for a form save are stored together as a single record in a new table called `cs_inform_form_record_history`. The answers themselves live inside a single JSON column on that record. This is significantly faster and more efficient, but it means the data layout is different.

What This Means for Your Reports

If your BI tool or reporting queries read directly from `cs_inform_response`, you will see a data split:

- **Old form submissions** (before this update): Still visible in `cs_inform_response`
- **New form submissions** (after this update): Stored in `cs_inform_form_record_history` as a JSON document, not as row-per-answer

Without any action, reports reading from `cs_inform_response` will appear to stop receiving new data after this update is deployed.

> **Earlier note about a `cs_inform_response_current` view:** Some earlier communication mentioned a compatibility view that would re-expand the JSON server-side. That approach has been dropped. The recommended pattern is now to read the history table directly and **handle the JSON decoding on the receiving end** (your BI pipeline, ETL job, or warehouse). This keeps load off the application database and lets you shape the data however your downstream tools need it.

The Solution — Read the History Table, Decode JSON Downstream

The new source of truth for form responses is the `cs_inform_form_record_history` table. Each row represents a single save of a single form record, with all answers bundled into the `responses_json` column.

Your BI / ETL pipeline should:

1. Query `cs_inform_form_record_history` (filtered by `is_latest = 1` if you only want the most recent state of each form), plus
2. Query `cs_inform_response` (filtered by `migrated_to_history = 0`) for any form records that haven't yet been snapshotted into the history table, then
3. **Decode** `responses_json` client-side (in your ETL job, BI tool's transformation step, or warehouse) to produce row-per-answer output if that's what your downstream consumers expect.

Most modern BI tools (Power BI, Tableau, Looker, Fabric, Snowflake, BigQuery, Databricks, dbt, Python/pandas, etc.) can parse and expand JSON natively. You no longer need the database to do it for you.

`cs_inform_form_record_history` Schema

Column	Description
`id`	Snapshot row identifier.
`form_record_id`	Links to `cs_inform_form_record` (the form submission).
`responses_json`	JSON document containing every answer for this save. See the structure below.
`schema_version`	Version of the JSON shape — currently `1`.
`submission_iteration`	How many times this form has been saved (1 = first save, increments on each subsequent save).
`status`	Status of the form record at the moment of this save.
`submission_method`	How the form was submitted. See values below.
`submission_request_id`	Optional request identifier supplied by the submitting client.

`trigger_event`	What caused this snapshot (e.g. `manual_save`, `status_change`, `backfill`).
`is_latest`	`1` for the most recent snapshot per `form_record_id`, `0` for superseded snapshots. Filter on this for current state.
`created_at`	When this snapshot was saved.
`created_by`	User ID who saved it.

`submission_method` values

Value	Meaning
`WEB_POST`	Standard form submission via the web interface
`WEB_AUTOSAVE`	Automatic save triggered while editing
`backfill_job`	Snapshot was created by migrating an older `cs_inform_response` record into the new history format

> Note for BI pipelines: `backfill_job` rows are historical data. They represent real form submissions and should be included in reports. The `created_at` value reflects when the original response was recorded.

`responses_json` Structure

Each `responses_json` document looks like this:

```
{
  "responses": {
    "a1b2c3d4-e5f6-7890-abcd-ef0123456789": {
      "value": "Yes",
      "sequence_number": 1,
      "question_version_id": 4711
    },
    "f0e9d8c7-b6a5-4321-fedc-ba9876543210-1": {
      "value": "Row 1 cell A",
      "sequence_number": 1,
      "question_version_id": 4823
    },
    "f0e9d8c7-b6a5-4321-fedc-ba9876543210-2": {
      "value": "Row 2 cell A",
      "sequence_number": 2,
      "question_version_id": 4823
    },
    "0c1b2a3d-9876-5432-10fe-dcba98765432": {
      "value": "[\"Option A\",\"Option B\"]",
```

```
      "sequence_number": 1,
      "question_version_id": 4925
    }
  },
  "submission_iteration": 3,
  "inline_element_counts": { "...": "..." }
}
```

Key things to know:

- Each key under `responses` is the question's UUID. For repeating / table-type questions, the key is suffixed with `-N` where `N` is the row's sequence number (e.g. `<uuid>-1`, `<uuid>-2`).
- `value` is always a string. Checkbox / multi-select questions store a JSON-encoded array as a string (e.g. `"[\\"Option A\\",\\"Option B\\"]"`). Your pipeline should detect a leading `[` and parse the inner JSON for those rows.
- `question_version_id` joins back to `cs_inform_form_question_version` if you need the question text, type, configuration, etc.
- `sequence_number` is the row index for repeating / table questions (always `1` for simple questions).
- `submission_iteration` and `inline_element_counts` are document-level fields, not per-answer.

Recommended Queries

Latest state of every form record (history-sourced)

```
SELECT
    h.form_record_id,
    h.id              AS history_snapshot_id,
    h.responses_json,
    h.submission_iteration,
    h.submission_method,
    h.created_at,
    h.created_by
FROM cs_inform_form_record_history h
WHERE h.is_latest = 1
  AND h.created_at >= '2024-01-01'
  AND h.created_at <  '2024-02-01';
```

Then unpack `responses_json` in your BI tool / ETL step.

Legacy responses not yet snapshotted

```
SELECT *
FROM cs_inform_response
WHERE migrated_to_history = 0
```

```
  AND deleted = 0
  AND created_at >= '2024-01-01'
  AND created_at <  '2024-02-01';
```

The `migrated_to_history` flag is maintained automatically: it stays `0` for legacy rows that still represent the source of truth, and is set to `1` once a snapshot in `cs_inform_form_record_history` exists for that form record. Filtering on this flag prevents double-counting once historical data is fully migrated.

Single form record — pulling all answers

```
SELECT responses_json
FROM cs_inform_form_record_history
WHERE form_record_id = 12345
  AND is_latest = 1;
```

(Falling back to `cs_inform_response WHERE form_record_id = 12345 AND migrated_to_history = 0` if no history snapshot exists yet.)

Unpacking `responses_json` Downstream

Once the JSON is in your warehouse / BI tool, expand it into row-per-answer. Pseudocode:

```
for each history_row in cs_inform_form_record_history:
    payload = json_parse(history_row.responses_json)
    for response_key, response in payload.responses.items():
        emit {
            form_record_id:        history_row.form_record_id,
            history_snapshot_id:   history_row.id,
            question_version_id:   response.question_version_id,
            sequence_number:       response.sequence_number,
            value:                 response.value,                    # may
be a JSON array string for checkboxes
            submission_method:     history_row.submission_method,
            submission_iteration:  history_row.submission_iteration,
            created_at:            history_row.created_at,
            created_by:            history_row.created_by
        }
```

Concrete examples in common tools:

- **dbt / Snowflake / BigQuery / Databricks SQL** — use `LATERAL FLATTEN`, `UNNEST`, or `LATERAL VIEW EXPLODE` over `JSON_EXTRACT(responses_json, '$.responses')`.
- **Power BI / Power Query** — use *Parse JSON* on `responses_json`, then *Expand to Rows* on the `responses` record.

- Python / pandas — `pd.json_normalize(json.loads(row['responses_json'])['responses'])`.
- SQL Server / Postgres / MariaDB — use `OPENJSON` / `jsonb_each` / `JSON_TABLE` over `responses_json`.

Known Differences from the Original `cs_inform_response` Table

Scenario	Original `cs_inform_response`	New `cs_inform_form_record_history`
Granularity	One row per answer	One row per form save (whole form bundled in `responses_json`)
Checkbox / multi-select	One row per selected option	A single `value` containing a JSON array string, e.g. `"[\\"Option A\\",\\"Option B\\"]"` — explode downstream
Per-answer `id` / `uuid`	Always populated	Not present — there is no row-level identifier per answer in the JSON
Identifying the latest state	Re-query the whole record	Filter on `is_latest = 1`
Joining answers to questions	`form_question_version_id` column	`responses_json.responses[*].question_version_id`
Repeating / table rows	Each row has its own `sequence_number`	Encoded in the response key suffix (`<uuid>-N`) and in `sequence_number` inside the JSON object

If your reports join or filter on a per-answer `id`, you will need to switch to joining on `form_record_id` + `question_version_id` (and `sequence_number` for table questions) instead.

Query Performance & Best Practices

The history table is much smaller than the legacy `cs_inform_response` table (one row per save instead of dozens), but it still pays to query it carefully.

Always use a date range filter

The single most important performance recommendation is to filter by `created_at`:

```
SELECT *
FROM cs_inform_form_record_history
WHERE is_latest = 1
  AND created_at >= '2024-01-01'
  AND created_at <  '2024-02-01';
```

Always use the `is_latest` flag if you want current state

```
WHERE is_latest = 1
```

This avoids scanning superseded snapshots — there's a dedicated index on `(is_latest, created_at)` for exactly this pattern.

For legacy data, use `migrated_to_history = 0`

```
WHERE migrated_to_history = 0
  AND deleted = 0
```

There's a dedicated index on `(migrated_to_history, deleted)` for this pattern. Filtering on the flag is much cheaper than the older `NOT EXISTS` style anti-join.

Extract data in chunks

For large historical extracts, pull one month at a time rather than the entire dataset in a single query. This keeps each query fast and prevents timeouts.

Decode JSON downstream, not in the source database

Decoding `responses_json` inside the source database puts load on the production application. Where possible, pull the raw JSON across the wire and let your warehouse / BI engine expand it — those systems are designed for that workload.

Expected query times

Query type	Expected performance
Single form record (`form_record_id = ?` and `is_latest = 1`)	Instant (< 1 second)
One month of latest snapshots (`is_latest = 1` + `created_at` range)	Fast (seconds)

One year of latest snapshots	Moderate (tens of seconds, depending on volume)
Full unfiltered extract	Slow — extract in monthly chunks instead

---

When Is the New Table Created?

`cs_inform_form_record_history` and the supporting `is_latest` / `migrated_to_history` flags are created automatically as part of the Ideagen EHS Core database migration that runs when this update is deployed to your environment. No manual action is required from you.

What Happens to the Old Data?

Nothing is deleted. All existing data in `cs_inform_response` remains intact and continues to be the source of truth for any form record that has **not** yet been snapshotted into `cs_inform_form_record_history` (those rows have `migrated_to_history = 0`).

A background migration process runs periodically to roll older records from `cs_inform_response` into `cs_inform_form_record_history`. As it does so, the corresponding `cs_inform_response` rows are marked with `migrated_to_history = 1`, and the matching history snapshot is marked `submission_method = 'backfill_job'`. Your reports should filter using both flags (`is_latest = 1` on history, `migrated_to_history = 0` on legacy) to get a complete and non-overlapping picture.

Frequently Asked Questions

Do I need to refresh or re-run anything to keep the data current?
No. `cs_inform_form_record_history` is a regular table that is written to in real time by the application. Every time your report runs, it sees the latest snapshots.

Will my existing reports break after this update?
Only if they query `cs_inform_response` directly without accounting for the new storage location. If you update your ETL / BI queries to read from `cs_inform_form_record_history` (for new submissions) and `cs_inform_response WHERE migrated_to_history = 0` (for legacy ones), and decode the JSON downstream, your reports will continue working without interruption.

Can I still query `cs_inform_response` directly?

Yes. The original table is still there and the data it contains has not changed. Just keep in mind that new submissions don't land there, and rows are progressively flagged with `migrated_to_history = 1` as their data is captured in the history table.

Why do some snapshots have `submission_method = 'backfill_job'`?

These snapshots were created by migrating older `cs_inform_response` records into the new history format. They represent real form submissions and should be treated the same as any other row. The `created_at` timestamp reflects the original submission time.

Why is my query slow even after switching to the history table?

The most common causes are a missing date filter, a missing `is_latest = 1` filter, or decoding the JSON server-side. Add a `WHERE created_at >= '...' AND created_at < '...' AND is_latest = 1` clause, and prefer to expand `responses_json` in your downstream warehouse / BI engine. See the Query Performance & Best Practices section above.

What about the `cs_inform_response_current` view I saw mentioned previously?

That view has been retired in favour of the simpler direct-table-plus-client-side-JSON approach. If any existing pipeline still references the view, replace it with the queries shown above. The view will be dropped automatically as part of the same database migration that ships this update.

Who do I contact if I have questions?

Please contact your Ideagen EHS Core account manager or support team.