**Fragment**
Easy, alternative soft deletion: `deleted_record_insert`

**Published**
December 28, 2022

Find me on Twitter at **@brandur**.

# Easy, alternative soft deletion: `deleted_record_insert`

A few months back I wrote *Soft deletion probably isn't worth it* (referring to the traditional strategy of putting a `deleted_at` column in each table), an assertion that I still stand behind. I've spent the time migrating our code away from `deleted_at`, and we're now at the point where it's only left on a couple core tables where we want to retain deleted records for an exceptionally long time for debugging purposes.

Nearer to the end of the article I suggest an alternative to `deleted_at` called `deleted_record`, a separate schemaless table that gets a full dump of deleted data, but which doesn't interfere with mainline code (no need to include a `deleted_at IS NULL` predicate in every live query, no foreign key problems), and without the expectation that it'll be used to undelete data (which probably wouldn't work for `deleted_at` anyway).

```
CREATE TABLE deleted_record (
    id uuid PRIMARY KEY DEFAULT gen_ulid(),
    data jsonb NOT NULL,
    deleted_at timestamptz NOT NULL DEFAULT current_timestamp,
    object_id uuid NOT NULL,
    table_name varchar(200) NOT NULL,
    updated_at timestamptz NOT NULL DEFAULT current_timestamp
);
```

Previously, I'd suggested manually writing `deleted_record` into each deletion query, but we've seen found a much cleaner way to do it. Here's a function which will generically insert a deleted record from any source table:

```
CREATE FUNCTION deleted_record_insert() RETURNS trigger
    LANGUAGE plpgsql
AS $$
    BEGIN
        EXECUTE 'INSERT INTO deleted_record (data, object_id, table_name) VALUES ($1, $2, $3)'
        USING to_jsonb(OLD.*), OLD.id, TG_TABLE_NAME;

        RETURN OLD;
    END;
$$;
```

Invoke it as an `AFTER DELETE` trigger on any table for which you want to retain soft deletion records:

```
CREATE TRIGGER deleted_record_insert AFTER DELETE ON credit
    FOR EACH ROW EXECUTE FUNCTION deleted_record_insert();
CREATE TRIGGER deleted_record_insert AFTER DELETE ON discount
    FOR EACH ROW EXECUTE FUNCTION deleted_record_insert();
CREATE TRIGGER deleted_record_insert AFTER DELETE ON invoice
    FOR EACH ROW EXECUTE FUNCTION deleted_record_insert();
```

Speaking from 30,000 feet, programming is all about tradeoffs. However, this is one of those rare places where as far as I can tell the cost/benefit skew is so disproportionate that the common platitude falls flat.

Since introducing this pattern months ago I haven't detected a single problem as it's happily worked away in the background without issue and there hasn't been a moment where I've found myself wishing that I had `deleted_at` back. During this time we've undoubtedly saved ourselves from dozens of bugs and countless hours of debugging time as people accidentally omit `deleted_at IS NULL` from production and analytical queries.

A++ programming pattern. Would implement again.

**Fragment**
Easy, alternative soft deletion: `deleted_record_insert`

**Published**
December 28, 2022

Find me on Twitter at **@brandur**.

Did I make a mistake? Please consider sending a pull request.