Database normalization

Use the `space_raw` table for this introduction. It is installed along with a clean installation of the hydenv database. If the table is not found, run:

```
python -m hydenv examples space
```

In case you don't want to run the SQL yourself, add the `--normalize` flag to the CLI call above and the CLI will normalize automatically.

Below, you will find the SQL queries used in the video, followed by a summary of the lessons learned.

SQL commands in the video

Normalization 02

2:07:

```sql
-- get all unique company names
SELECT DISTINCT company_name FROM space_raw;
```

4:52:

```sql
-- use a row count as the new id for primary key
SELECT
  row_number() OVER (ORDER BY company_name ASC) AS company_id,
  names.company_name
FROM (
    SELECT DISTINCT company_name FROM space_raw
) as names;
```

9:15:

```sql
-- create the companies table
DROP TABLE IF EXISTS companies CASCADE;
CREATE TABLE companies AS
SELECT
  row_number() OVER (ORDER BY company_name) AS company_id,
  t.company_name
FROM (SELECT DISTINCT company_name FROM space_raw) t;
ALTER TABLE companies ADD CONSTRAINT pkey_companies
    PRIMARY KEY (company_id);
SELECT * FROM companies;
```

Normalization 03

2:55:

```sql
-- get only the rocket name from the detail attribute
SELECT split_part(detail, ' | ', 1) as rocket_name FROM space_raw;
```

6:20

```
SELECT
    split_part(detail, ' | ', 1) as rocket_name,
    CASE WHEN status_rocket='StatusActive' THEN
        true
    ELSE false END as is_active
 FROM space_raw
```

8:31

```
-- create the rocket table
DROP TABLE IF EXISTS rockets CASCADE;
CREATE TABLE rockets AS
SELECT
    row_number() OVER (ORDER BY t.rocket_name) AS rocket_id,
    t.rocket_name, is_active
FROM
(
    SELECT DISTINCT
        split_part(detail, ' | ', 1) as rocket_name,
        CASE WHEN status_rocket='StatusActive' THEN
            true
        ELSE false END as is_active
    FROM space_raw
) t;
ALTER TABLE rockets ADD CONSTRAINT pkey_rockets PRIMARY KEY (rocket_id);
SELECT * FROM rockets LIMIT 15;
```

Normalization 04

1:49:

```
SELECT DISTINCT
  split_part(location, ', ', 3) AS part_3,
  split_part(location, ', ', 4) AS part_4
FROM space_raw
```

3:21:

```
SELECT DISTINCT
CASE WHEN split_part(location, ', ', 4) ='' THEN
        split_part(location, ', ', 3)
    ELSE
        split_part(location, ', ', 4)
    END as country
FROM space_raw
```

6:29:

```
DROP TABLE IF EXISTS locations CASCADE;
CREATE TABLE locations AS
SELECT
    row_number() OVER (ORDER BY identifier) AS location_id, t.*
FROM
(
    SELECT DISTINCT
        split_part(location, ', ', 1) as identifier,
        split_part(location, ', ', 2) as location_name,
        CASE WHEN split_part(location, ', ', 4) = '' THEN
            null
        ELSE
            split_part(location, ', ', 3)
        END as state,
        CASE WHEN split_part(location, ', ', 4) = '' THEN
            split_part(location, ', ', 3)
        ELSE
            split_part(location, ', ', 4)
        END AS country
    FROM space_raw
) t;
ALTER TABLE locations ADD CONSTRAINT pkey_locations
    PRIMARY KEY (location_id);
SELECT * FROM locations;
```

Normalization 05

11:09

```
DROP TABLE IF EXISTS spaces CASCADE;
CREATE TABLE spaces AS
SELECT
    id, datum,
    (
        SELECT company_id FROM companies
        WHERE company_name=space_raw.company_name
    ) as company_id,
    (
        SELECT location_id FROM locations WHERE
            identifier=split_part(location, ', ', 1) AND
            location_name=split_part(location, ', ', 2)
    ) as location_id,
    (
        SELECT rocket_id FROM rockets
        WHERE rocket_name=split_part(detail, ' | ', 1)
    ) as rocket_id,
```

```
    split_part(detail, ' | ', 2) as mission_detail,
    status_mission
FROM space_raw;
ALTER TABLE spaces ADD CONSTRAINT pkey_space PRIMARY KEY (id);
ALTER TABLE spaces ADD CONSTRAINT fkey_space_location
    FOREIGN KEY (location_id) REFERENCES locations (location_id);
ALTER TABLE spaces ADD CONSTRAINT fkey_space_rocket
    FOREIGN KEY (rocket_id) REFERENCES rockets (rocket_id);
ALTER TABLE spaces ADD CONSTRAINT fkey_space_company
    FOREIGN KEY (company_id) REFERENCES companies (company_id);
SELECT * FROM spaces LIMIT 15;
```

Summary

Why normalizing?

- the relational concept alone does not make a good database
- a database is only as useful as the structure it represents
- the data model is way more important than the choice of RDBMS
- The structure should be as simple as possible while being as complex as necessary

What is DB normalization?

- The overall goal is to avoid data inconsistencies and make the schema flexible
- The database is split up into different tables, each one representing only one *topic*
- Avoid duplicated *values* in the database
- non-normalized databases are usually screwed up, when editing data
- There are 5 rules of normalization, of which three will be covered
- The five rules will be applied one after another

First normal form

The database is in first normal form when all data ranges are **atomic**.

This is important to make the attributes query-able and avoid inconsistencies.

non-atomic:

customer_id	purchase
3	3948573 - socks , Bakers, San Francisco
4	3902938 - pullover , Bakers, San Francisco
5	3948573 - sock , Marcy's, Los Angeles

atomic:

customer_id	art_id	art_name	store_name	store_location
3	3948573	socks	Bakers	San Francisco
4	3902938	pullover	Bakers	San Francisco
5	3948573	sock	Marcy's	Los Angeles

Second normal form

The database is in second normal form when it's in fist normal form and any non-prime attribute is fully dependent on the primary key.

Simpler: Put everything that has duplicated values into its own table, if possible.

customer_id	art_id	store_id
3	3948573	1
4	3902938	1
5	3948573	2

art_id	name
3948573	socks
3902938	pullover

store_id	name	location
1	Bakers	San Francisco
2	Marcy's	Los Angeles

Third normal form

A database is in third normal form when it's in second normal form and no non-prime attribute is transitive dependent on any other non-prime attribute.

Simpler: Everything that's not a primary or foreign key must not depend on anything else than a primary key.

Consider:

store_id	name	location
1	Bakers	San Francisco
2	Marcy's	Los Angeles

Here, `location` is transitive dependent on `name` and not on `the store_id`.

Normalized:

store_id	name	location_id
1	Bakers	1
2	Marcy's	2

location_id	name
1	San Francisco
2	Los Angeles

Then, if we open more stores:

store_id	name	location_id
1	Bakers	1
2	Marcy's	2
3	Bakers	2
4	Tech Market	2