Databases for Data Science

Lecture 05 · 2022-09-13

Today

- More about constraints
- Third normal form
- Database programming
  - Stored procedures
  - Functions
  - Triggers (time permitting)

We'll be doing some more exercises along the way :)

Constraints

You can (and generally should) name your constraints:

```
CREATE TABLE timespans (
    timespan_id SERIAL PRIMARY KEY,

    -- Column with column constraint
    start_date date
        NOT NULL
        CONSTRAINT start_after_2020 CHECK(start_date >= '2021-01-01'),

    end_date date,

    -- Table constraint
    CONSTRAINT dates_make_sense CHECK (end_date IS NULL OR start_date <= end_date)
);
```

Constraints

You can use `ALTER TABLE` to add constraints:

```
ALTER TABLE course_catalog
    ADD CONSTRAINT pandemic CHECK (remote_option OR term NOT IN ('SP20','FA20','SP21')),
    ADD CONSTRAINT courses_unique UNIQUE (course_number, term),

    -- foreign keys are special
    ADD FOREIGN KEY (course_id) REFERENCES course_info(course_id),

    -- NOT NULL is special
    ALTER COLUMN term SET NOT NULL;
```

Individual Exercise: in your personal DB, add some constraints to the `student` and `financial_aid` tables.

Constraints

Use `\d my_table` to view the active constraints.

- If you didn't name a constraint, you can see its generated name here.

Constraints can be dropped:

```
ALTER TABLE course_catalog
    DROP CONSTRAINT pandemic,

    -- NOT NULL is special
    ALTER COLUMN term DROP NOT NULL;
```

Constraints

If we're using `CREATE TABLE my_table AS SELECT ...`, how can we assign constraints?

```
CREATE TABLE financial_aid AS
    SELECT
        student_id,
        (1000.0 * student.gpa)::MONEY
    FROM
        student;

ALTER TABLE my_table
    ADD CONSTRAINT
```

Constraints

Can't do it in the `CREATE` command; instead, use a transaction:

```
BEGIN; -- Open a transaction
CREATE TABLE financial_aid_2 AS
    SELECT
        student_id,
        (1000.0 * student.gpa)::MONEY as scholarship
    FROM student;

ALTER TABLE financial_aid_2
    ADD CONSTRAINT minimum_scholarship CHECK (scholarship > '$500'),
    ADD FOREIGN KEY (student_id) REFERENCES student(student_id)
        ON DELETE CASCADE; -- You can specifty deletion behavior
COMMIT; -- Close the transaction
```

Review: Normal forms

1NF

Every attribute (column) represents a single value.

- No lists
- No compound attributes

Every entity (row) row is unique.

2NF

Full functional dependencies:

Any column that is not part of a key must not be functionally dependent on *part* of any key.

- If you have a single key that is atomic, you get this for free!

Review: Normal Forms

2NF

Full functional dependencies:

Any column that is not part of a key must not be functionally dependent on *part* of a key.

term	coursenum	instructor	department
FA22	IDC 5130	W. Corning	Data Science
SP21	ART 3200	K. Anderson	Art

- What's the primary key?
- Is this in 2NF? If not, how could we normalize it?

Review: Functional Dependencies

Each value on the left-hand side has *at most one* associated right-hand side.

$$x \rightarrow y$$

$$ssn \rightarrow (first\_name, last\_name, dob)$$

$$student\_id \longleftrightarrow email$$

Third Normal Form

2NF, and:

Every non-key column is functionally dependent *only* on the key(s).

"The key, the whole key, and nothing but the key."

- No transitive dependencies.
  - e.g. `product_id -> (manufacturer, SKU) -> product_name`
  - e.g. `bookingnumber -> soid -> dob`

Third Normal Form

3NF

Every non-key column is functionally dependent *only* on the key(s).

- No transitive dependencies.
  - e.g. `product_id -> (manufacturer, SKU) -> product_name`
  - e.g. `bookingnumber -> soid -> dob`

How can we test for this? (Logically, not with SQL)

Third Normal Form

3NF

Every non-key column is functionally dependent *only* on the key(s).

- No transitive dependencies.
  - e.g. `product_id -> (manufacturer, SKU) -> product_name`
  - e.g. `bookingnumber -> soid -> dob`

How can we test for this? (Logically, not with SQL)

- If we changed the value in *this* column (or set of columns), would something else need to be different?

Recap

1NF

Every column is atomic; every row is distinct.

2NF

Any column that is not part of a key must not be functionally dependent on *part* of any key.

3NF

Every non-key column is functionally dependent *only* on the key(s).

Third Normal Form

Group Exercise:

I've copied a subset of `homelessness` to `mini_homelessness`.

Pick a one-word name for your group (or use your own name).

Copy `mini_homelessness` to a new database:

- `create database lec6_[GROUP NAME] with template mini_homelessness;`

Enter your new database: `\c lec6_[GROUP NAME]`

Third Normal Form

2NF
Any column that is not part of a key must not be functionally dependent on *part* of any key.

3NF
Every non-key column is functionally dependent *only* on the key(s).

Group Exercise: Normalize your database into 3NF.

- Design your normalized schema; draw a rough ER-like diagram.
  - Clearly identify primary and foreign keys.
- Write an SQL transaction to create the tables for your design and populate them with data.
  - Add any relevant constraints.

Working in a Normalized Database

Normalization has advantages, but also creates complexity.

- You'll need to write more joins!

Individual exercise:
In your normalized database, create a view that reproduces the original `bookings` table.

Individual exercise:
In your normalized database, write a query to produce the `chargetype`, `charge`, and `releasecode` for every cannabis-related arrest.

Working in a Normalized Database

Most of the time, external records (CSV files, etc) will not be normalized.

Inserting into a normalized DB can be tricky.

- Consider the normalized `orders` database you made yesterday.

  How would you insert data from the CSV?

Working in a Normalized Database

Temporary tables are a useful tool for this.

```
BEGIN; -- Start transaction
CREATE TEMPORARY TABLE temp_import (
    email text,
    -- etc
);
COPY temp_import FROM file.csv WITH CSV HEADER;

-- Create rows for any new users
INSERT INTO user (email, name)
SELECT (email, name)
FROM temp_import
WHERE email NOT IN (SELECT email FROM user);

--- etc
END; -- Commit transaction, dropping the temp table
```

Working in a Normalized Database

Group Exercise:

Finish and run the import code on the previous slide.

You can use your own database design from yesterday, or copy mine.
(run the SQL file at `/usr/share/databases/wcorning/orders-database.sql`)

Programming in Databases

Sometimes we'll be running the same kind of command many times.

In such cases, it's useful to identify the varying parameters and store the command in a way that is easy to invoke.

Likewise, we may be using the same expression (formula) repeatedly; we'd like to write it down somewhere stable.

Stored Procedures

```
CREATE OR REPLACE PROCEDURE insert_student(student_name text)
LANGUAGE PLPGSQL -- This isn't plain SQL!
AS $$

DECLARE -- variable declarations
email TEXT := replace(student_name, ' ', '') || '@ncf.edu';

BEGIN -- start of procedural section

INSERT INTO student(name, email)
VALUES (student_name, email);

END; -- end of procedural section
$$;
```

```
CALL insert_student('Xavier Barnes');
```

Stored procedures

```
CREATE OR REPLACE PROCEDURE insert_student(student_name text)
LANGUAGE PLPGSQL
AS $$
DECLARE email TEXT := replace(student_name, ' ', '') || '@ncf.edu';
BEGIN
    INSERT INTO student(name, email)
    VALUES (student_name, email);
END; -- end of procedural section
$$;
```

PL/pgSQL is a language that extends SQL with procedural programming features.

Exercise: Make your import code into a stored procedure. It should take the filename as its argument.

Functions

```
CREATE FUNCTION increment(a int, b int default 1)
RETURNS INT
AS $$
BEGIN
    RETURN a + b;
END;
$$ LANGUAGE plpgsql;
```

```
select increment(5);
```

Triggers

[Stretch topic]

Next week

- Wrap up SQL
  - Revisit indexes and query plans
  - Discuss optimization
  - Consider other SQL databases
- Begin with MongoDB

Before you go

Food logistics.

Yesterday: $108.70 total
→ $8.00 / student

Today: $139.69 total
→ ???

- what sounds fair?