Databases for Data Science

Lecture 04 · 2022-09-09

Today

- Writing data: create, insert, update, delete
- Transactions
- Keys and normal forms
- Table constraints
- Python drivers for PostgreSQL

Creating a new database

- `[user@cs1 ~]$ createdb my_database` creates a db named `my_database`
  - `[user@cs1 ~]$ createdb` creates a db named `user`
- `user=# CREATE DATABASE my_database`

Deleting a database

- `[user@cs1 ~] dropdb my_database`
- `user=# DROP DATABASE my_database`

Creating a table

```
CREATE TABLE my_table (
    some_column INT,
    some_column VARCHAR(255)
    some_column TEXT,
    some_column BOOLEAN,
    some_column NUMERIC
);
```

Creating a table

```
CREATE TABLE student (
    student_id   INT,
    name         VARCHAR(255),
    bio          TEXT,
    on_campus    BOOLEAN,
    gpa          NUMERIC
);
```

Inserting values

```
INSERT INTO student VALUES
    (1, 'Alex', '...lives in Sarasota...', false, 3.2),
    (2, 'Blake', 'Aspiring data scientist', TRUE, 3.5);
```

Creating table from query

```
CREATE TABLE financial_aid AS
    SELECT
        student_id,                                  -- column from student
        CAST('2021-08-01' AS DATE) AS start_date,    -- convert this string to a date
        (2000 * gpa / 4.0)::MONEY  AS scholarship    -- another syntax for casting
    FROM student;
```

Inserting data from a query

```
INSERT INTO financial_aid (
    SELECT
        student_id,
        CURRENT_DATE, -- PostgreSQL function
        500           -- Note that this will be interpreted as a MONEY value
    FROM student
    WHERE on_campus
);
```

Deleting values

```
DELETE FROM financial_aid WHERE start_date < '2021-08-01';
```

Deleting values

- What would happen if we ran `DELETE FROM financial_aid;`?

Dropping tables

```sql
-- make a table
CREATE TABLE example AS SELECT * FROM student;

-- drop (delete) the table
DROP TABLE example;
```

Importing data

Recall that you can copy data into a database:

```
\copy table_name FROM /location/on/disk WITH CSV HEADER
```

Exercise

Create a table named `orders` for the data in `/usr/share/databases/example/orders.csv`.

Copy that file into your table.

Updating rows

We can change the values that are already present in a table.

```
-- Adjust GPA
UPDATE student
SET gpa=4.0
WHERE name='Blake';
```

Exercise

Covid time! Everybody moves off-campus; update the `students` table.

Altering tables

Whoops, we forgot last names.

```
ALTER TABLE student
    RENAME name first_name
    ADD last_name varchar(255);

UPDATE student SET last_name = 'Ample' WHERE first_name = 'Alex';
UPDATE student SET last_name = 'Blark' WHERE first_name = 'Blake';
```

Transactions

If some part of an `UPDATE` query fails, no changes will be made.

What if we're running a series of `UPDATE`s, and something goes wrong midway through?

Transactions

If some part of an `UPDATE` query fails, no changes will be made.

What if we're running a series of `UPDATE`s, and something goes wrong midway through?

Transactions

Transaction: a group of operations that either succeeds completely or has no effect.

```
-- Start a transaction
BEGIN;

-- Empty the financial_aid table
DELETE FROM financial_aid;

-- This won't work!
INSERT INTO financial_aid
    VALUES (3, CURRENT_DATE, 1/0);

-- Finalize the transaction (causing a rollback due to error)
COMMIT;
```

ACID Properties

- **Atomicity**: A transaction will succeed or fail as a single "operation"; it will not have partial effects.
- **Consistency**: A transaction must leave the database in a valid state.
- **Isolation**: Transactions will not interfere with each other.
- **Durability**: Once a transaction completes, its effects will not be lost.

A database system is **ACID Compliant** if its transactions satisfy these properties.

Transactions

Command	Meaning
`BEGIN;`	Begin setting up a transaction
`COMMIT;`	Finish setup and attempt to execute transaction
`ROLLBACK;`	Cancel the transaction

`SAVEPOINT my_save;`	Create savepoint
`ROLLBACK TO my_save;`	Cancel everything after the savepoint

Views

We created `financial_aid` with scholarships based on `gpa`, but now it's out of date.

What if we wanted everyone's scholarship to reflect their *current* gpa?

View

Views allow you to create a virtual table with the behavior of some query.

Every time you reference a view, its underlying query will be run.

```
CREATE VIEW merit_based_aid AS (
    SELECT
        student_id,
        (1000 * gpa / 4.0) as scholarship
    FROM
        student
);
```

Views

Exercise

Create a view that shows the off-campus students and their current total scholarship.

Python and Postgres

(VSCode)

Normalization

Why not have one big table?

Why not have copies of a column in multiple tables?

Why not have duplicate rows in the same table?

Keys

How can we uniquely identify data in our databases?

- What have we seen in `homelessness`, and what were the issues with this?

Keys

- **Primary key**: a piece of information that acts as a unique identifier for every entity (row in the table)
- **Foreign key**: a primary key from another table, used as a "link" between entities

We may also be concerned with...

- **Composite key**: a set of fields that collectively act as a key
- **Candidate key**: something that *could* be a primary key

First normal form

Every row is unique, and represents a single value.

- No duplicate rows.
- No columns containing lists.

(How could we check this?)

First normal form

Every row is unique, and represents a single value.

- No duplicate rows.
- No columns containing lists.

Group exercise: put `orders` into first normal form.

Second normal form

First normal form, *and:*

Every non-key column depends on the entire primary key, not a part; there are no partial dependencies.

Group exercise: put `orders` into second normal form.

Third normal form

Second normal form, *and:*

Every non-key column depends *directly* on the primary key, and not indirectly on another non-key column. There are no transitive dependencies.

Group exercise: put `orders` into third normal form.