NYU | Center for Data Science

Week 02.4: Relational databases

DS-GA 1004: Big Data

Announcements

- **Lab 0 is due Friday (02/04)**
- Lab 1 (SQL) starts this week
  - Due in ~2 weeks: 02/18
- Office hours on an altered schedule this week + next

Previously...

- File systems for data storage
- Limitations:
  - No constraints on data formats / validation
  - Does not support content-based querying
  - No management of concurrent access

This week

- Relational databases
- SQL
- Transactions

slido

How much experience do you have with SQL/RDBMS prior to this course?

ⓘ Start presenting to display the poll results on this slide.

Recap: the relational model

- **High-level**: tables of data that you're probably used to
  - Spreadsheets, dataframes, numerical arrays, etc.

- Each column represents a **set** of possible values (numbers, strings, etc)

- A ***relation*** over sets $A_1, A_2, A_3, \ldots, A_n$ is a **subset** of their cartesian product
  - $R \subseteq A_1 \times A_2 \times A_3 \times \ldots \times A_n$
  - The **rows** of the table are elements of R, also known as ***tuples***
  - $(a_1, a_2, \ldots, a_n) \in R \Rightarrow a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n$

Example: dinosaurs

- A_1 = {*s* | *s* is a string}
  A_2 = {“Jurassic”, “Cretaceous”, “Devonian”, “Triassic”, …}
  A_3 = {“Carnivore”, “Herbivore”, “Omnivore”, …}
  A_4 = {False, True}

- Any A_i could be finite or infinite

- $R \subseteq A_1 \times A_2 \times A_3 \times A_4$ need not contain all combinations!

Species	Era	Diet	Awesome
T. Rex	Cretaceous	Carnivore	True
Stegosaurus	Jurassic	Herbivore	True
Ankylosaurus	Cretaceous	Herbivore	False

slido

If A has 5 elements and B has 3 elements, and A∩B=⌀, how many possible relations exist using sets A and B?

ⓘ Start presenting to display the poll results on this slide.

slido

[2] If A has 5 elements and B has 3 elements, and A∩B=⌀, how many possible relations exist using sets A and B?

ⓘ Start presenting to display the poll results on this slide.

Solution: counting relations

- Count the combinations:

  $|A| = 5, |B| = 3 \Rightarrow |A \times B| = 15$

- # Relations = # Possible subsets of the combinations

  $|2^{A \times B}| = 2^{15}$

- Order matters! $A \times B \neq B \times A$. We need to count both!

  $2^{15} + 2^{15}$

- But now we've double-counted the empty relation $\varnothing \in 2^{A \times B}$ and $\varnothing \in 2^{B \times A}$

  $\Rightarrow 2^{15} + 2^{15} - 1 = 2^{16} - 1$

Example

- A = {1, 2, 3, 4, 5} B = {😀, 😐, ☹}
- (1, 😀) ∈ A×B
  (😀, 1) ∉ A×B
- Order of the columns matters!

Schemas

id	Species	Era	Diet	Awesome
1	T. Rex	Cretaceous	Carnivore	True
2	Stegosaurus	Jurassic	Herbivore	True
3	Ankylosaurus	Cretaceous	Herbivore	False
4	Homer	Boomer	Donuts	False

- A relation is defined by a **schema:**

Dinosaur(id: int, Species: string, Era: string, Diet: string, Awesome: boolean)

- **Any** tuple (int, string, string, string, boolean) is valid under this schema
  - ⇒ Schemas enforce type (syntax), but not semantics!

Schemas can be hard to design!

- Imagine making a schema to track customers

  Customer(id: int, lastname: string, firstname: string)

- **Are all strings valid as names**?

slido

What constraints could you add to the name field of our customer database?

ⓘ Start presenting to display the poll results on this slide.

This can go wrong very easily...

- Length constraints are problematic in both directions
- So are character set constraints (accents, spaces, etc)
- Search and linkage are difficult if the data must be modified to fit schema
- This affects different populations disproportionately. **Be careful!**

Structured Query Language

- SQL is the language we use to talk to databases
  - Not a procedural language like Python or C
  - **Declarative**: state what you want, not how to compute it
- Think of it more like a **protocol** than a programming language
- SQL is an ANSI standard, but different implementations each have quirks
  - **MySQL** vs **Postgres** vs **SQLite** vs **MSSQL** …

Use your RDBMS library to sanitize queries

```
db.execute(“ SELECT * FROM Dinosaur
             WHERE species = ‘%s’” % name)
```

name variable becomes **part of** the query code!

```
db.execute(“ SELECT * FROM Dinosaur
             WHERE species = ‘?’”, name)
```

name variable is a **parameter** to the query code!

Python + sqlite3 example

Indexing

id	Name	Country	Street	Zip
1	T. Rex	US	5th Ave.	10003
2	Stegosaurus	US	8th St.	10004
3	Ankylosaurus	CA	Spadina Ave.	M5T 3A5

- An **index** is a data structure over one or more columns that can accelerate queries

- Example:
  - A table that has a few distinct values repeated millions of times
  - And you frequently want all rows with exactly one given value
  - It might be faster to store a mapping **value → rows** than to search each row independently

When to index?

- When data is **read more often** than written
- When queries are **predictable**
- When queries rely on a **small number of attributes**
- **Remember**: you can always add or delete indices later

ACID

Atomicity	Operations are all-or-nothing (No partial updates; operations bundled in **transactions**)
Consistency	Transactions move from one **valid** state to another
Isolation	Concurrent operations do not depend on **order** of execution
Durability	Completed transactions are **permanent** (usually implemented by flushing to disk before completion)

Transactions

- When modifying tables, wrap query statements in

  BEGIN TRANSACTION; [queries]; **COMMIT**;

  or

  BEGIN TRANSACTION; [queries]; **ROLLBACK**;

- Different DBMS use slightly different syntax (**START**, **BEGIN**)

- If a query **fails** mid-transaction, uncommitted changes will be **abandoned**
  - ⇒ DB is always left in a **valid state**

Aside: transactions example

- SQL transactions are kind of like “try … except” blocks in Python/Java/etc
- It’s helpful to think of SQL interactions as a conversation or **session**
  - Each command executes and either succeeds or fails
- If you’re performing multiple queries, and one of them fails for some reason, you can **roll back** to the state of the database at the beginning
- If everything completes successfully, you can **commit** the transaction

Example

id	balance (≥ 0)
1	$10
2	$20

- Imagine transferring $20 from account id=2 to account id=1
  - “balance” column cannot go negative
- This is done with two update queries in a transaction:
  - BEGIN TRANSACTION
    - UPDATE account SET balance=balance-20 WHERE id=2
    - UPDATE account SET balance=balance+20 WHERE id=1
  - COMMIT
- These need to either both happen or neither happens

Adapted from https://www.sqlitetutorial.net/sqlite-transaction/

Example

id	balance (≥ 0)
1	$10
2	$20

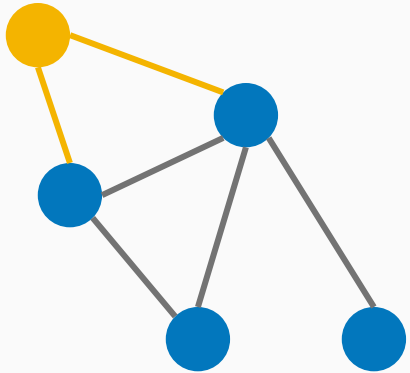
- If we instead wanted to transfer $20 from id=1 to id=2, this would violate the schema constraint “balance >= 0”.

- This is done with two update queries:
  - BEGIN TRANSACTION
    - UPDATE account SET balance=balance-20 WHERE id=1 ← fails!
    - UPDATE account SET balance=balance+20 WHERE id=2
  - ROLLBACK

- Second query is not committed - all changes are reverted!

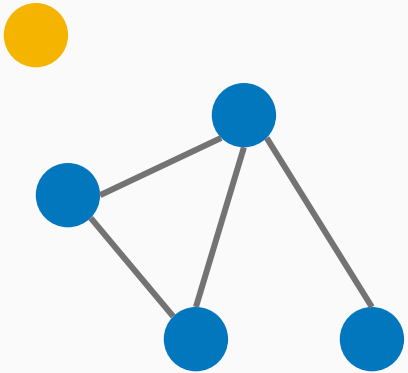
FS drawbacks: consistency!

- What if you want to impose **constraints** on your data?
- Example:
  - Guaranteeing that a graph is **connected**
  - Vertices stored in **nodes.dat**
  - Edges stored in **edges.dat**
  - What if you want to add a **vertex** and two **edges**?



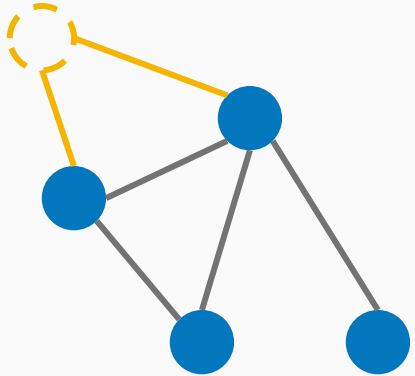
FS drawbacks: consistency!

- What if you want to impose **constraints** on your data?
- Example:
  - Guaranteeing that a graph is **connected**
  - Vertices stored in **nodes.dat**
  - Edges stored in **edges.dat**
  - What if you want to add a **vertex** and two **edges**?
- Add vertex first: graph is **disconnected**



FS drawbacks: consistency!

- What if you want to impose **constraints** on your data?
- Example:
  - Guaranteeing that a graph is **connected**
  - Vertices stored in **nodes.dat**
  - Edges stored in **edges.dat**
  - What if you want to add a **vertex** and two **edges**?
- Add edges first: **edges** are **invalid**



slido

Which property would be most useful for fixing our connected graph problem?

ⓘ Start presenting to display the poll results on this slide.

slido

[2] Which property would be most useful for fixing our connected graph problem?

ⓘ Start presenting to display the poll results on this slide.

Next week

- Distributed computation with Map-Reduce
- Reading:
  - First: Dean & Ghemawat
  - Second: DeWitt & Stonebraker

slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.