Introduction To Relational Data Models

Lecture 3, Chapter 2

John Connor

August 28, 2018

The goal of the next several lectures is to

1. develop a method of representing data as sets of tuples;
2. create a mathematical language which will allow us to query these sets;
3. translate this framework into languages, algorithms, and data-structures which can actually be implemented in software.

Why Do We Use Set Theory For Databases?

Recall from Lecture 1 that we wish to store and query "instances" of "structures". For example,

```
struct Movie {
    char title[128];
    int year;
    int length;
    char genre[128];
}

Movie m1 = {
    "Gone With the Wind", 1939, 231, "drama"
};
Movie m2 = { "Star Wars", 1977, 124, "scifi" };
```

Why Do We Use Set Theory For Databases?

Recall from Lecture 1 that we wish to store and query "instances" of "structures". For example,

```
struct Movie {
    char title[128];
    int year;
    int length;
    char genre[128];
}

Movie m1 = {
    "Gone With the Wind", 1939, 231, "drama"
};
Movie m2 = { "Star Wars", 1977, 124, "scifi" };
```

In C, we might store these instances in some data-structure which we could then query via loops and if statements.

Why Do We Use Set Theory For Databases?

However, we don’t want to mathematically model the “low level” details of C-queries. We only want to model the questions and answers themselves, not the method by which questions are transformed into answers.

Why Do We Use Set Theory For Databases?

However, we don't want to mathematically model the "low level" details of C-queries. We only want to model the questions and answers themselves, not the method by which questions are transformed into answers.

To do this we represent the `struct Movie` instances as tuples, and we "store" our tuples in sets. For example,

$$\{(\text{Gone With the Wind}, 1939, 231, \text{drama}),\ (\text{Star Wars}, 1977, 124, \text{scifi})\}$$

Why Do We Use Set Theory For Databases?

We need to model `struct Movie` as well as its instances. We do this by creating a **schema**.

Why Do We Use Set Theory For Databases?

We need to model `struct Movie` as well as its instances. We do this by creating a **schema**.

A schema is just a tuple of **attributes**, and an attribute is just a pair of the form $(string, type)$.

Why Do We Use Set Theory For Databases?

We need to model `struct Movie` as well as its instances. We do this by creating a **schema**.
A schema is just a tuple of **attributes**, and an attribute is just a pair of the form $(string, type)$.

The schema for `struct Movie` is then

$$((\text{“title”}, string), (\text{“year”}, int), (\text{“length”}, int) (\text{“genre”}, string))$$

Why Do We Use Set Theory For Databases?

We need to model `struct Movie` as well as its instances. We do this by creating a **schema**.
A schema is just a tuple of **attributes**, and an attribute is just a pair of the form $(string, type)$.

The schema for `struct Movie` is then

$$((\text{“title”}, \mathit{string}), (\text{“year”}, \mathit{int}), (\text{“length”}, \mathit{int}) (\text{“genre”}, \mathit{string}))$$

A set of tuples along with their schema is called a **relation** (or sometimes relation instance.)

The Table Representation

We will often make use of a notational convention where a relation is depicted as a table. For example,

Table: Example relation for the Movie schema.

title	**year**	**length**	**genre**
Gone With the Wind	1939	231	drama
Star Wars	1977	124	sciFi
Wayne's World	1992	95	comedy

The Table Representation

We will often make use of a notational convention where a relation is depicted as a table. For example,

Table: Example relation for the Movie schema.

title	**year**	**length**	**genre**
Gone With the Wind	1939	231	drama
Star Wars	1977	124	sciFi
Wayne's World	1992	95	comedy

The header of the table is made from the names in the schema, and the rows of the table are just the tuples in our relation.

Keys

Can two different movies have the same title?

Keys

Can two different movies have the same title?

. . . Of course, it happens all the time!

Keys

Can two different movies have the same title?

. . . Of course, it happens all the time!

Can there be two different movies with the same title released in the same year?

Keys

Can two different movies have the same title?

. . . Of course, it happens all the time!

Can there be two different movies with the same title released in the same year?

. . . Admittedly, this seems very unlikely.

Keys

We can constrain the `Movies` schema so that no two movies have the same name and the same year attributes by asserting that $\{$`title, year`$\}$ is a **key**.

Keys

We can constrain the `Movies` schema so that no two movies have the same name and the same year attributes by asserting that {`title, year`} is a **key**.

A set of attributes is called a key if no two tuples in the relation instance can have the same values for all elements of the key.

Compact Schema Representation

Sometimes we will need to name and present a schema in a compact way.

In these cases we will just write the schema name as a prefix to a tuple containing the attribute names. If we need to emphasize a key, we will underline the names forming it. For example,

$$\texttt{Movie(}\underline{\texttt{title}}\texttt{, }\underline{\texttt{year}}\texttt{, length, genre)}$$

Non-Relational Data Models (Just to Be Thorough)

Data models generally fall into one of two categories, relational and semi-structured.

Non-Relational Data Models (Just to Be Thorough)

Data models generally fall into one of two categories, relational and semi-structured.
We will focus on relational models, but some examples of semi-structured models are

1. XML
2. JSON

Non-Relational Data Models (Just to Be Thorough)

Data models generally fall into one of two categories, relational and semi-structured.
We will focus on relational models, but some examples of semi-structured models are

1. XML
2. JSON

We may take a look at JSON towards the end of the semester.