Databases

CS3100 Fall 2019

Review

Previously

- Graph Search.

This lecture

- Connections between SQL and Prolog

Relational Databases

- A database is a store of facts.
- A relation database is organized on the principles of relational model
  - Consists of one or more tables with rows and named columns
- A table schema captures
  - the column names
  - types over values
  - any constraints on values in each column
  - relationship between between columns across different tables
- Structured Query Language (SQL)
  - A standard language used to read and write to relational databases.

IMDB database

- For this section, we will focus on a small slice of the IMDB database.
- The database contains information about the movies directed by a few directors.
- The database `imdb_small.db` can be explored using `sqlite` in terminal.

IMDB tables

```
CREATE TABLE tPeople (
  person_id varchar primary key,
  name varchar,
  born integer);

CREATE TABLE tTitles (
  title_id varchar primary key,
  title varchar,
  premiered integer,
  runtime_minutes integer,
  genres varchar);
);
```

IMDB tables

```
CREATE TABLE tDirectedBy (
  title_id varchar,
  person_id varchar,
  primary key (title_id, person_id));

CREATE TABLE tRatings (
  title_id VARCHAR PRIMARY KEY,
  rating INTEGER,
  votes INTEGER
);
```

Representing relational tables in Prolog

```
+--------------------------+
|     SQL      |   Prolog  |
+--------------------------+
| tables       | predicate |
| rows         | fact      |
| column names | -         |
| schema       | -         |
+--------------------------+
```

tPeople table in Prolog

```
/* tPeople(person_id, name, born). */
tPeople(nm0634240,"Christopher Nolan",1970).
tPeople(nm0000217,"Martin Scorsese",1942).
tPeople(nm0000233,"Quentin Tarantino",1963).
tPeople(nm0000229,"Steven Spielberg",1946).
```

Let's load all the data into Prolog from the file `imdb_small.pl` .

In [1]:

```
?- [imdb_small].
```

```
true.
```

Select rows in SQL

Get me all the rows from the `tPeople` table.

```
sqlite> select * from tPeople;
nm0634240|Christopher Nolan|1970
nm0000217|Martin Scorsese|1942
nm0000233|Quentin Tarantino|1963
nm0000229|Steven Spielberg|1946
```

Select rows in Prolog

In prolog, the query is represented by the same predicate that defines the table.

In [2]:

```
?- tPeople(PersonId,Name,Born).
```

```
PersonId = nm0634240, Born = 1970, Name = Christopher Nolan ;
PersonId = nm0000217, Born = 1942, Name = Martin Scorsese ;
PersonId = nm0000233, Born = 1963, Name = Quentin Tarantino ;
PersonId = nm0000229, Born = 1946, Name = Steven Spielberg .
```

Select with filter

Get me all the information about `Christopher Nolan` from tPeople table.

```
sqlite> select * from tPeople where name="Christopher Nolan";
nm0634240|Christopher Nolan|1970
```

In [3]:

```
?- tPeople(PersonId,Name,Born), Name="Christopher Nolan".
```

```
PersonId = nm0634240, Born = 1970, Name = Christopher Nolan .
```

Select with filter

```
sqlite> select * from tPeople where born > 1960;
nm0634240|Christopher Nolan|1970
nm0000233|Quentin Tarantino|1963
```

In [4]:

```
?- tPeople(PersonId,Name,Born), Born > 1960.
```

```
PersonId = nm0634240, Born = 1970, Name = Christopher Nolan ;
PersonId = nm0000233, Born = 1963, Name = Quentin Tarantino .
```

Projection

Projection is act of choosing a subset of columns from the table.

```
sqlite> select Name,Born from tPeople where born > 1960;
Christopher Nolan|1970
Quentin Tarantino|1963
```

In [5]:

```
?- tPeople(_,Name,Born), Born > 1960.
```

```
Born = 1970, Name = Christopher Nolan ;
Born = 1963, Name = Quentin Tarantino .
```

DirectedBy

`tDirectBy` table contains movies associates a director with the movie that they directed.

```
sqlite> select * from tDirectedBy limit 5;
tt0053416|nm0000217
tt0054670|nm0000229
tt0054857|nm0000229
tt0057680|nm0000217
tt0058242|nm0000217
```

In [6]:

```
?- tDirectedBy(TitleId,PersonId) {5}.
```

```
PersonId = nm0000217, TitleId = tt0053416 ;
PersonId = nm0000229, TitleId = tt0054670 ;
PersonId = nm0000229, TitleId = tt0054857 ;
PersonId = nm0000217, TitleId = tt0057680 ;
PersonId = nm0000217, TitleId = tt0058242 .
```

How do we get this information in human readable form?

Joins

We do this through joins. Let's begin with a quick primer on joins.

Given

```
A = {1,2,3,4,5}
B = {3,4,5,6,7}
```

- An **inner join** is said to be $A \cap B$ = `{3,4,5}` .
- A **left outer join** is said to be $A \cup (A \cap B)$ = `{1,2,3,4,5}` .
- A **right outer join** is said to be $(A \cap B) \cup B$ = `{3,4,5,6,7}` .
- A **full outer join** is said to be $A \cup B \cup (A \cap B)$ = `{1,2,3,3,4,5,6,7}` .

The outer joins seems to serve no purpose as the term $A \cap B$ is included in the other terms.

This get more interesting with additional columns.

Joins

Similar to previous example, consider

```
Persons = {(1,"Spielberg"),(2,"Nolan")}
Directed = {(2,"The Dark Knight"), (3,"Taxi Driver")}
```

An X join of Persons and Directed on the `person_id` field, selecting name and movie returns,

- X = Inner, `{("Nolan","The Dark Knight)}`
- X = Left outer, `{("Spielberg",null),("Nolan","The Dark Knight)}`
- X = Right outer, `{("Nolan","The Dark Knight),(null,"Taxi Driver"}`
- X = Full outer, `{("Spieldberg",null),("Nolan","The Dark Knight),(null,"Taxi Driver"}`

In this lecture, we will only focus on inner joins.

Joins

In order to illustrate other kinds joins in Prolog, let's consider this simple database from the earlier example.

In [7]:

```
sPerson(1,"Spielberg").
sPerson(2,"Nolan").
sDirected(2,"The Dark Knight").
sDirected(3,"Taxi Driver").
```

```
Added 4 clauses(s).
```

Inner Join

In [8]:

```
innerJoin(Title,Director) :- sPerson(PersonId,Director), sDirected(PersonId,Title).
```

```
Added 1 clauses(s).
```

In [9]:

```
?- innerJoin(Title,Director).
```

```
Director = Nolan, Title = The Dark Knight .
```

Left outer Join

In [10]:

```
leftOuterJoin(Title,Director) :- innerJoin(Title,Director).
leftOuterJoin(null,Director) :- sPerson(PersonId,Director), not(sDirected(PersonId,T
```

```
Added 2 clauses(s).
```

In [11]:

```
?- leftOuterJoin(Title,Director).
```

```
Director = Nolan, Title = The Dark Knight ;
Director = Spielberg, Title = null .
```

Right outer join

In [12]:

```
rightOuterJoin(Title,Director) :- innerJoin(Title,Director).
rightOuterJoin(Title,null) :- sDirected(PersonId,Title), not(sPerson(PersonId,Direct
```

```
Added 2 clauses(s).
```

In [13]:

```
?- rightOuterJoin(Title,Director).
```

```
Director = Nolan, Title = The Dark Knight ;
Director = null, Title = Taxi Driver .
```

Exercise

Encode full outer join in Prolog and run it on this example.

Joins

Select the movie title and the corresponding ratings.

```
sqlite> select tTitles.title, rating from tRatings
           inner join tTitles on tTitles.title_id = tRatings.title_id limit
10;
Firelight|5.6
Who's That Knocking at My Door|6.7
Street Scenes|6.4
Boxcar Bertha|6
Mean Streets|7.3
Alice Doesn't Live Here Anymore|7.3
The Sugarland Express|6.8
Jaws|8
Taxi Driver|8.3
Close Encounters of the Third Kind|7.6
```

Joins

Select the movie title and the corresponding ratings.

In [14]:

```
ratings(Title,Rating) :- tTitles(TitleId,Title,_,_,_), tRatings(TitleId,Rating,_).
```

```
Added 1 clauses(s).
```

In [15]:

```
?- ratings(Title,Rating).
```

```
Rating = 5.6, Title = Firelight ;
Rating = 6.7, Title = Who's That Knocking at My Door ;
Rating = 6.4, Title = Street Scenes ;
Rating = 6, Title = Boxcar Bertha ;
Rating = 7.3, Title = Mean Streets ;
Rating = 7.3, Title = Alice Doesn't Live Here Anymore ;
Rating = 6.8, Title = The Sugarland Express ;
Rating = 8, Title = Jaws ;
Rating = 8.3, Title = Taxi Driver ;
Rating = 7.6, Title = Close Encounters of the Third Kind .
```

Joins

What is the rating for the movie "Jaws"?

```
sqlite> select rating from tRatings inner join tTitles on tTitles.title_id
= tRatings.title_id where tTitles.title = "Jaws";
8
```

In [16]:

```
?- ratings("Jaws",Rating).
```

```
Rating = 8 .
```

Join on 3 tables

Get all the movies directed by Nolan.

```
sqlite> select tTitles.title, tPeople.name from
           tTitles inner join tDirectedBy on tTitles.title_id = tDirectedBy
.title_id
           inner join tPeople on tPeople.person_id = tDirectedBy.person_id
           where tPeople.name = "Christopher Nolan";
Following|Christopher Nolan
Memento|Christopher Nolan
Insomnia|Christopher Nolan
Batman Begins|Christopher Nolan
The Dark Knight|Christopher Nolan
The Prestige|Christopher Nolan
Interstellar|Christopher Nolan
The Dark Knight Rises|Christopher Nolan
Inception|Christopher Nolan
Dunkirk|Christopher Nolan
Tenet|Christopher Nolan
```

Join on 3-tables

In [17]:

```
directed(Director,Title) :- tTitles(TitleId,Title,_,_,_), tPeople(PersonId,Director,
```

```
Added 1 clauses(s).
```

In [18]:

```
?- directed("Christopher Nolan",Title).
```

```
Title = Following ;
Title = Memento ;
Title = Insomnia ;
Title = Batman Begins ;
Title = The Dark Knight ;
Title = The Prestige ;
Title = Interstellar ;
Title = The Dark Knight Rises ;
Title = Inception ;
Title = Dunkirk .
```

Count

How many movies has Nolan directed?

Use the `count` function.

```
sqlite> select count(*) from tTitles inner join tDirectedBy on tTitles.titl
```

```
e_id = tDirectedBy.title_id inner join tPeople on tPeople.person_id = tDire
ctedBy.person_id where tPeople.name = "Christopher Nolan";
11
```

Count

Use the built-in clause `findall/3` .

The built-in predicate `findall(+Template, +Goal, -List)` is used to collect a list `List` of all the items `Template` that satisfy some goal `Goal` .

In [19]:

```
directedList(D,L) :- findall(F,directed(D,F),L).
```

```
Added 1 clauses(s).
```

In [20]:

```
numDirected(D,N) :- directedList(D,L), length(L,N).
```

```
Added 1 clauses(s).
```

In [21]:

```
?- numDirected("Christopher Nolan",N).
```

```
N = 11 .
```

Exercise: Try to build `findall/3` your self?

Avg

What is the average rating for a Spielberg movie?

```
sqlite> select avg(rating) from tRatings
          inner join tDirectedBy on tDirectedBy.title_id = tRatings.title_i
d
          inner join tPeople on tPeople.person_id = tDirectedBy.person_id
          where tPeople.name = "Steven Spielberg";
7.31515151515151
```

Avg

What is the average rating for a Spielberg movie?

In [22]:

```
ratingOf(Ratings,Name) :- tRatings(TitleId,Ratings,_), tDirectedBy(TitleId,PersonId
                          tPeople(PersonId,Name,_).
```

```
Added 1 clauses(s).
```

In [23]:

```
sum([H],H).
sum([H|T],N) :- sum(T,M), N is M+H.
average(L,A) :- sum(L,S), length(L,N), A is S / N.
```

```
Added 3 clauses(s).
```

In [24]:

```
averageRatingOf(Name,A) :- tPeople(_,Name,_),findall(Rating,ratingOf(Rating,Name),L
```

```
Added 1 clauses(s).
```

In [25]:

```
?- averageRatingOf("Steven Spielberg",A).
```

```
A = 7.31515151515 .
```

Avg

What is the average rating for each of the directors?

```
sqlite> select avg(rating),tPeople.name from tRatings
          inner join tDirectedBy on tDirectedBy.title_id = tRatings.title_i
d
          inner join tPeople on tPeople.person_id = tDirectedBy.person_id
          group by tPeople.name;
8.25|Christopher Nolan
7.48|Martin Scorsese
7.95714285714286|Quentin Tarantino
7.31515151515151|Steven Spielberg
```

In [26]:

```
?- averageRatingOf(X,A).
```

```
A = 8.25, X = Christopher Nolan ;
A = 7.48, X = Martin Scorsese ;
A = 7.95714285714, X = Quentin Tarantino ;
A = 7.31515151515, X = Steven Spielberg .
```

Upcoming movies

- SQL uses `NULL` to represent missing values.
- In the `tTitles` table, `NULL` is used for upcoming movies.

```
sqlite> select * from tTitles where premiered is NULL;
tt1594575|Untitled George Gershwin Project|||Biography,Drama,Music
tt3675680|The Kidnapping of Edgardo Mortara|||Drama,History
tt5537002|Killers of the Flower Moon|||Crime,Drama,History
tt7428530|Roosevelt|||Biography,Drama,History
tt7713358|Untitled Star Trek Project|||Action,Adventure,Sci-Fi
tt8295436|Blackhawk|||Action,Adventure,War
tt8430788|Untitled Ulysses S. Grant Project|||Drama,War
```

Upcoming movies

In the Prolog version, we use the constant `null` to represent missing values.

In [27]:

```
?- tTitles(TitleId,Name,null,null,Genres).
```

```
Genres = Biography,Drama,Music, Name = Untitled George Gershwin Projec
t, TitleId = tt1594575 ;
Genres = Drama,History, Name = The Kidnapping of Edgardo Mortara, Titl
eId = tt3675680 ;
Genres = Crime,Drama,History, Name = Killers of the Flower Moon, Title
Id = tt5537002 ;
Genres = Biography,Drama,History, Name = Roosevelt, TitleId = tt742853
0 ;
Genres = Action,Adventure,Sci-Fi, Name = Untitled Star Trek Project, T
itleId = tt7713358 ;
Genres = Action,Adventure,War, Name = Blackhawk, TitleId = tt8295436 ;
Genres = Drama,War, Name = Untitled Ulysses S. Grant Project, TitleId
= tt8430788 .
```

Dealing with NULL

We will have to deal with null values specially as their semantics is what we choose it to be.

Get me all the movies that are released on likely to be released on or after 2019.

```
sqlite> select * from tTitles where premiered >= 2019;
tt1302006|The Irishman|2019|209|Biography,Crime,Drama
tt1462764|Untitled Indiana Jones Project|2021||Action,Adventure
tt3581652|West Side Story|2020||Crime,Drama,Musical
tt6723592|Tenet|2020||Action,Drama,Thriller
tt7131622|Once Upon a Time... in Hollywood|2019|161|Comedy,Drama
tt9577852|Rolling Thunder Revue: A Bob Dylan Story by Martin Scorsese|2019|
142|Biography,Documentary,Music
```

- Does not return movies for which released date is not set.
  - We also want potential upcoming movies.

Dealing with NULL

```
sqlite> select * from tTitles where premiered >= 2019 or premiered is null;
tt1302006|The Irishman|2019|209|Biography,Crime,Drama
tt1462764|Untitled Indiana Jones Project|2021||Action,Adventure
tt1594575|Untitled George Gershwin Project|||Biography,Drama,Music
tt3581652|West Side Story|2020||Crime,Drama,Musical
tt3675680|The Kidnapping of Edgardo Mortara|||Drama,History
tt5537002|Killers of the Flower Moon|||Crime,Drama,History
tt6723592|Tenet|2020||Action,Drama,Thriller
tt7131622|Once Upon a Time... in Hollywood|2019|161|Comedy,Drama
tt7428530|Roosevelt|||Biography,Drama,History
tt7713358|Untitled Star Trek Project|||Action,Adventure,Sci-Fi
tt8295436|Blackhawk|||Action,Adventure,War
tt8430788|Untitled Ulysses S. Grant Project|||Drama,War
tt9577852|Rolling Thunder Revue: A Bob Dylan Story by Martin Scorsese|2019|
142|Biography,Documentary,Music
```

Dealing with NULL

In [28]:

```
premieredAfter(Name,_) :- tTitles(_,Name,null,_,_).
premieredAfter(Name,D) :- tTitles(_,Name,Premiered,_,_), not(Premiered=null), Premie
```

```
Added 2 clauses(s).
```

In [29]:

```
?- premieredAfter(M,2019) {20}.
```

```
M = Untitled George Gershwin Project ;
M = The Kidnapping of Edgardo Mortara ;
M = Killers of the Flower Moon ;
M = Roosevelt ;
M = Untitled Star Trek Project ;
M = Blackhawk ;
M = Untitled Ulysses S. Grant Project ;
M = The Irishman ;
M = Untitled Indiana Jones Project ;
M = West Side Story ;
M = Tenet ;
M = Once Upon a Time... in Hollywood ;
M = Rolling Thunder Revue: A Bob Dylan Story by Martin Scorsese .
```

Recursive Queries

SQL has no way to express recursive queries. In Prolog, this is quite natural.

- Let's define a predicate `hop` between two titles either
  - if they were released in the same year, or
  - if they were directed by the same person.

In [30]:

```
hop(TitleId1,TitleId2) :-
  tTitles(TitleId1,_,Premiered,_,_),
  tTitles(TitleId2,_,Premiered,_,_).
hop(TitleId1,TitleId2) :-
  tDirectedBy(TitleId1,Person),
  tDirectedBy(TitleId2,Person).
```

```
Added 2 clauses(s).
```

Recursive Queries

Let's define `reachable` between two titles if one can be reached from the other by one or more hops.

In [31]:

```
reachable(TitleId1,TitleId2,_) :- hop(TitleId1,TitleId2).
reachable(TitleId1,TitleId2,Visited) :-
  hop(TitleId1,TitleId3), \+member(TitleId3,Visited), reachable(TitleId3,TitleId2,[T
```

```
Added 2 clauses(s).
```

Recursive Queries

Finally, let's define `connected` on movie titles, if the title_ids are reachable.

In [32]:

```
connected(Title1,Title2) :-
  tTitles(TitleId1,Title1,_,_,_), tTitles(TitleId2,Title2,_,_,_),
  reachable(TitleId1,TitleId2,[TitleId1]).
```

```
Added 1 clauses(s).
```

In [33]:

```
?- connected("Dunkirk","Jaws") {1}.
```

```
true.
```

Fin.