Lecture 4
Introduction to Queries to Knowledge Bases
Semantic Parsing for Database Queries

Very important source of data
Lots of knowledge in structured data (public & private)

- **Correctness**
  Can we generate the correct queries?
- **Coverage**
  Can we handle what people ask? Head vs. tail queries?
- **Completeness**
  Can we understand all possible db queries?
- **Cost**
  What is the cost of training data acquisition?

The 4 Cs!
Outline

1. Are Siri, Alexa, and Google assistant good enough? (Traditional approach)
2. Why do we care about databases?
3. Key Insight: Schema-based approach
4. Query representation: ThingTalk
5. Getting used to natural language as database queries
6. Queries factored into operators and domain knowledge
7. An Overview to Training
Asking Siri, Alexa, Google for Help with Restaurants

“Find me a Chinese restaurant”
Asking Siri, Alexa, Google for Help with Restaurants

“Find me a Chinese restaurant with at least 4.5 stars”
Asking Siri, Alexa, Google for Help with Restaurants

“Find me a Chinese restaurant with at least 4.5 stars”

Alexa

Google Assistant

Siri

Genie

I see Veggie Garden, Redwood Bistro, and Blend Eatery. They’re restaurants rated 4.5 star with Chinese food.
## Genie vs Commercial Assistants on Long-Tail Questions

<table>
<thead>
<tr>
<th>Examples of Long-Tail Questions</th>
<th>Alexa</th>
<th>Google</th>
<th>Siri</th>
<th>Genie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show me restaurants rated at least 4 stars with at least 100 reviews</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Show restaurants in San Francisco rated higher than 4.5</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>What is the highest rated Chinese restaurant near Stanford?</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>How far is the closest 4 star restaurant?</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Find a W3C employee that went to Oxford</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Who worked for Google and lives in Palo Alto?</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Who graduated from Stanford and won a Nobel prize?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Who worked for at least 3 companies?</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Show me hotels with checkout time later than 12PM</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Which hotel has a pool in this area?</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Quiz

• Is it important to do the compound statements?

• Is it important to handle the long tail?
Old Assumption

No real users $\Rightarrow$ No NLP
Traditional Approach: Usage Driven

- Collect data from the wild

Collect questions from users

Developers annotate questions with logical form

**Question**
Show me Chinese restaurants in Stanford.

**Logical form**
@restaurant() filter servesCuisine =~ "chinese " &&
location = new Location("Stanford")

one training example
Usage Driven & A Logical Form Representation

- Represent sentence in a logical form
  - Captures what the user is saying
  - Logical form is not executable
- Interpret the logical form to perform the action
AMRL: Alexa Meaning Representation Language

Manual annotation in AMRL

Get me an upscale restaurant
Search for an upscale restaurant and then make a reservation for it
找一家高档餐厅，然后帮我预约

Possible Requests

Get me an upscale restaurant
What are the restaurants around here?
What is the best restaurant?
Search for Spanish restaurants
Search for Spanish restaurants open at 10pm

Domains

Restaurants
Music
Sports
News
IoT devices

Languages

English
Chinese
Hindi
Spanish
Arabic

Alexa: 10,000 employees
Traditional Approach: Usage Driven

- Collect data from the wild
  - Collect questions from users
    - Handles the head
    - Easy to see how simple natural languages translate into complex queries
    - Cannot cover the combinations
    - Not trying to be complete

- Developers annotate questions with logical form
  - Hard to annotate correctly
  - much harder than object recognition
  - High cost

But we need to understand sentences we have not seen before!
Outline

1. Are Siri, Alexa, and Google assistant good enough? (Traditional approach)
2. Why do we care about databases?
3. Key Insight: Schema-based approach
4. Query representation: ThingTalk
5. Getting used to natural language as database queries
6. Queries factored into operators and domain knowledge
7. An Overview to Training
What do people care about?
Public Live Knowledge Bases: Wikidata

- A community-driven knowledge graph
  - 100M items: each Wikipedia page has an item in Wikidata
  - 10K properties, 12B facts, 22K contributors
  - Represented in a semantic web, retrieved with SPARQL (Property, ID, value)

Who was Stanford University named after?

Stanford University (Q41506) item identifier
private research university located in Stanford, California, United States
Leland Stanford Junior University | University of Stanford | Stanford | The Farm | stanford.edu

property named after
value Leland Stanford, Jr

0 references
Example:
Which country has the most official languages?

SPARQL
SELECT DISTINCT ?uri WHERE {
    GROUP BY ?uri ORDER BY DESC(COUNT(?language)) LIMIT 1
}

ThingTalk
sort (count (official_language desc of @wd.country()))[1];
Who wrote Jurassic Park? Michael Crichton
That was such a good movie.
What was his first big movie? Andromeda Strain in 1971
And the second? Westworld in 1973
Westworld? I didn’t know that.
Who starred in it? Yul Brynner, Richard Benjamin
Public Knowledge Bases: Schema.org

- The web has a schema: **Schema.org**
  - Structure data to mark up web pages
  - Mainly used by search engines
  - It covers many domains, including restaurants, hotels, people, recipes, products, news ...

- Core: 792 types, 1447 properties
- + extensions

```html
<script type="application/ld+json">{
  @type: "restaurant",
  name: "The French Laundry",
  servesCuisine: "French",
  aggregateRating: {
    @type: "AggregateRating",
    reviewCount: 2527,
    ratingValue: 4.5
  }
}...
```

Schema.org markup on Yelp
Private Knowledge Bases

• Every company is a data company [Forbes 2018]
  • Grocery store: customers, stock info
  • Insurance: customers, plan information
  • Restaurant: menu, opening hours
• Challenges in data acquisition
  • Company proprietary or confidential information
  • User privacy
  • Limited resources to collect data for question answering

Databases: one of the most important CS abstractions
  Same code to handle knowledge for all companies

Can we write the same code to handle all natural language queries to DBs?
1. Are Siri, Alexa, and Google assistant good enough? (Traditional approach)
2. Why do we care about databases?
3. **Key Insight: Schema-based approach**
4. Query representation: ThingTalk
5. Getting used to natural language as database queries
6. Queries factored into operators and domain knowledge
7. An Overview to Training
Key Insights

To be truthful, natural language utterances need to be grounded with live databases

Databases are a key CS abstraction

- Same code to handle knowledge for all companies
- Companies just supply the schema

Is it possible that the same code can handle natural language queries to DBs?

- Companies just supply the schema
- DBs have a small common number of operators, which can be composed
- Use LLMs to provide domain terminology
## New Approach: Usage Driven

<table>
<thead>
<tr>
<th>Natural Language</th>
<th>Neural Semantic Parser</th>
<th>Database Query</th>
<th>Execution Database Queries</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Traditional: Usage Driven</th>
<th>New: Schema Driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start from the user</td>
<td>Go backwards.</td>
</tr>
<tr>
<td></td>
<td>Start from what the computer can do. (Completeness)</td>
</tr>
<tr>
<td>Non-executable logical form</td>
<td>Represent as executable queries</td>
</tr>
<tr>
<td></td>
<td>- No need to interpret a logical form</td>
</tr>
<tr>
<td></td>
<td>- End-to-end: No intermediate representation between human &amp; computer</td>
</tr>
<tr>
<td></td>
<td>- Independent of the source natural language (Supports multi-linguality)</td>
</tr>
<tr>
<td>Manually annotated usage-based training data</td>
<td>Synthesize a sample of all queries (with the help of LLMs)</td>
</tr>
<tr>
<td></td>
<td>+ a few shot of manual annotation</td>
</tr>
</tbody>
</table>
Quiz

• Schema-driven approach
  • Lots of correctly annotated data that are synthesized
  • Strengths?

• Weaknesses?
Quiz

• Is an executable representation suitable for all NLP problems?
1. Are Siri, Alexa, and Google assistant good enough? (Traditional approach)
2. Why do we care about databases?
3. Key Insight: Schema-based approach
4. **Query representation: ThingTalk**
5. Getting used to natural language as database queries
6. Queries factored into operators and domain knowledge
7. An Overview to Training
How Database Queries Work

• Relational databases
  • Query language: SQL
  • Relational algebra: 6 fundamental operations + derived operations
    • Selection (σ)
    • Projection (Π)
    • Union (U)
    • Set difference (−)
    • Cartesian product / Join (×)
    • Rename (ρ) -- not necessary in single commands in natural language
  • Same algebra works for all domains!
What is ThingTalk? And why ThingTalk?

Why can’t we just use SQL. After all, LLMs have seen SQL syntax.

• Main reason: SQL is used only for databases, not a general language
  • Typical programs require: SQL queries + a general programming language
• ThingTalk is much more expressive:
  • Database queries
  • API calls: e.g. Yelp restaurant is not a relational database, but an API call
  • Combinations of API and DB queries
  • Operations: Filters; Control flow: events and function composition
• ThingTalk has a compiler and run-time system
Example of a ThingTalk Statement

[when =>]? get, filter => do

“every day at 9am play songs by Taylor Swift”

attimer(time=9:00) => @com.spotify.song(), contains(artists, “taylor swift”) => @com.spotify.play(playable=id)

• **When**: when the command is to be executed – *when clause*
• **Get**: a query – *noun*
• **Do**: call an action – *verb*

Uniform syntax for APIs & DBs

Event-driven execution in 1 line. Each clause can be executed by a different internet resource. With authentication when needed. Most concise networking code.

Custom-generated filter if not supported by APIs
Other ThingTalk Advantages

• Extensibility
  • Access control, Satisfiability Modulo Theories (SMT) formulas
  • GUI operators, pushing buttons, typing in textboxes
• Designed for natural language translation
  • E.g. No variables
• Fully compositional: syntax can easily be composed (useful for program manipulation and synthesis)

ThingTalk syntax is easy for neural networks to learn. The actual domain names and fields are harder.
Let’s Focus on the Query Subset of ThingTalk

- Similar in power as core SQL
- Constructs matching the user’s mental model (implicit variables)

<table>
<thead>
<tr>
<th>Question</th>
<th>ThingTalk</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>the restaurant</td>
<td>@restaurant()</td>
<td>SELECT * FROM restaurant</td>
</tr>
<tr>
<td>menu of the restaurant</td>
<td>[menu] of @restaurant()</td>
<td>SELECT menu FROM restaurant</td>
</tr>
<tr>
<td>number of reviews of the restaurant</td>
<td>[count(reviews)] of @restaurant()</td>
<td>SELECT count(review.id) FROM restaurant JOIN review ON review.restaurant= restaurant.id GROUP by restaurant.id</td>
</tr>
<tr>
<td>menu and number of reviews of the restaurant</td>
<td>[menu, count(reviews)] of @restaurant()</td>
<td>SELECT count(review.id), restaurant.menu FROM restaurant JOIN review ON review.restaurant= restaurant.id GROUP by restaurant.id</td>
</tr>
</tbody>
</table>
Selection

Show me restaurants **in** Palo Alto

```javascript
@restaurant() filter geo == new Location("Palo Alto")
```

```javascript
table [filter filter]? 
```
Strong Typing

• All parameters and return values are strongly typed
  • Boolean, String, Number, Date, Time, Location
  • Measure (unit-parameter); e.g. C (Celsius), ft, ...
  • Currency (ISO-code parameter)
  • Entity (type-parameter): the value is the identifier of an object
    Type-parameter: picture, hashtag, username, path_name, url, phone_number, email_address, device, function
  • Enum (e.g. on, off)
  • Array
  • Compound (structure)
Filter Operators

• Typing enables filters automatically
  • Previous approach: possible expressions are designed domain by domain
• Type-specific operators:
  • Numeric: $\geq$, $\leq$
  • Array: contains, in_array
  • String: =~ (soft match), starts_with, ends_with, prefix_of, suffix_of
• Logical:
  • $\&\&$, $\|$, $!$
Outline

1. Are Siri, Alexa, and Google assistant good enough? (Traditional approach)
2. Why do we care about databases?
3. Key Insight: Schema-based approach
4. Query representation: ThingTalk
5. **Getting used to natural language as database queries**
6. Queries factored into operators and domain knowledge
7. An Overview to Training
Show me Chinese restaurants in Palo Alto
Show me Chinese restaurants in Palo Alto with at least 4.5 stars.
Show me Chinese restaurants in Palo Alto with at least 4.5 stars
Show me the address of Chinese restaurants in Palo Alto with at least 4.5 stars.
Show me the address of Chinese restaurants in Palo Alto with at least 4.5 stars.
Show me the address of Chinese restaurants in Palo Alto reviewed by Bob.
Show me the address of Chinese restaurants in Palo Alto reviewed by Bob

```
sort ( field asc|desc of table [filter filter]? )

[address] of
@restaurant() filter geo == new Location("Palo Alto")
&& servesCuisine =~ "Chinese"
&& contains(reviews, any(@review() filter author =~ "Bob" ))
```
Sorting

```
sort ( field asc|desc of table [filter filter]? )
```

```
sort ( rating desc of
    @restaurant() filter geo == new Location("Palo Alto")
    && servesCuisine =~ "Chinese"
    && contains(reviews, any(@review() filter author =~ "Bob" )))
```

Show me **top-rated** Chinese restaurants **in Palo Alto** reviewed by Bob
Joins

\[
\text{table [filter filter]? join table [filter filter]? [on filter]?}
\]

\[
\text{sort ( rating desc of @restaurant() filter geo == new Location(“Palo Alto”)}
\]
\[
\text{&& servesCuisine =~ “Chinese”}
\]
\[
\text{&& contains(reviews, any(@review() filter author =~ “Bob” )))}
\]

Show me top-rated Chinese restaurants in Palo Alto reviewed by Bob
Show me top-rated Chinese restaurants in Palo Alto with their reviews

table [filter filter]? join table [filter filter]? [on filter]?
	sort ( rating desc of 
@restaurant() filter geo == new Location("Palo Alto")
&& servesCuisine =~ "Chinese"
join @review() on contains(first.reviews, second.id))
### ThingTalk Language Summary (subset)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Filter operators</strong></td>
<td>and(,), or, not, &gt;=, &lt;=, contains, in_array, =~, starts_with, ends_with, prefix_of, suffix_of</td>
</tr>
<tr>
<td><strong>Selection</strong></td>
<td><strong>Atom filter</strong> rating &gt;= 4.5</td>
</tr>
<tr>
<td></td>
<td><strong>Computation filter</strong> count(reviews) &gt;= 100</td>
</tr>
<tr>
<td></td>
<td><strong>Subquery</strong> contains(reviews, any(@review() filter author =~ “Bob” ))</td>
</tr>
<tr>
<td></td>
<td><strong>Existential subquery</strong> any([speed] of @gps())</td>
</tr>
<tr>
<td><strong>Projection</strong></td>
<td>[address] of @restaurant()</td>
</tr>
<tr>
<td><strong>Boolean question</strong></td>
<td>[rating &gt;= 4.5] of @restaurant()</td>
</tr>
<tr>
<td><strong>Join</strong></td>
<td>@restaurant() join @review() on contains(first.reviews, second.id)</td>
</tr>
<tr>
<td><strong>Sorting &amp; Limiting</strong></td>
<td>sort (rating desc of @restaurant()) [1:5]</td>
</tr>
<tr>
<td><strong>Aggregation</strong></td>
<td>count(@restaurant()) avg(rating of @restaurant())</td>
</tr>
<tr>
<td><strong>Computation</strong></td>
<td>[distance(geo, $location)] of @restaurant()</td>
</tr>
</tbody>
</table>
Canonicalized ThingTalk

Sentences with the same meaning are represented with identical ThingTalk

“show me a restaurant that serves Italian food or serves Chinese food”

@restaurant(), servesCuisine =~ “Italian” ||
          servesCuisine =~ “Chinese”

"show me a restaurant that serves Italian or Chinese food”

@restaurant, in_array~(servesCuisine,
                   [“Italian”, “Chinese”])

• The two semantics are the same
• Always use the second one
• Filters are ordered alphabetically
Quiz

• What is the purpose of canonicalization?
Quiz

• What is the purpose of strong typing?

• Can ThingTalk answer everything the user wants to know?

• How do we prove that ThingTalk is good for semantic parsing?
Quiz

• ThingTalk has a formal grammar. What can we do with the grammar?
Quiz

• Will SQL Work for the Database Subset?
SQL will probably work well for single query commands
Wait till you see conversations!

Our goal: standardize ThingTalk.
Outline

1. Are Siri, Alexa, and Google assistant good enough? (Traditional approach)
2. Why do we care about databases?
3. Key Insight: Schema-based approach
4. Query representation: ThingTalk
5. Getting used to natural language as database queries
6. **Queries factored into operators and domain knowledge**
7. An Overview to Training
Database Abstraction

DB Queries factored into

Relational Operators
Implemented by
DB writers

Domain Knowledge: Schemas
Defined by
App developers

Natural language query factored into

σ, Π, U, −, ×

Natural language constructs

Restaurant
Name | Price | Cuisine | ...

User
Name | Email | Age | ...

City
Name | Geo | Country | ...

Product
Name | UPC | Price | ...
## Canonical NL Construct

<table>
<thead>
<tr>
<th>Relational Algebra</th>
<th>Canonical NL Template</th>
<th>Database Schema + Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection ((\sigma))</td>
<td>(&lt;\text{table}&gt; \text{ with } &lt;\text{param}&gt; \text{ equals } &lt;\text{value}&gt;)</td>
<td>(\text{restaurants} \text{ with cuisine equal to Chinese})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{users} \text{ with name equal to John})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{city with country equal to United States})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{product with price equal to $10})</td>
</tr>
<tr>
<td>Projection ((\Pi))</td>
<td>(&lt;\text{param}&gt; \text{ of } &lt;\text{table}&gt;)</td>
<td>(\text{menu of restaurant})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{name of user})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{country of city})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{price of product})</td>
</tr>
</tbody>
</table>

How many NL templates do we need?
Quiz

• How many natural language templates do we need?

• Given a ThingTalk program, can we generate the natural language equivalent?

• What can do we with it that?
# DB Constructs: Ground English to ThingTalk

Canonical English templates (grammar): covers all queries

<table>
<thead>
<tr>
<th>Operation</th>
<th>English Template</th>
<th>ThingTalk</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>table with fname equal to value</td>
<td>table, fname = value</td>
<td>restaurants with rating equal to 3</td>
</tr>
<tr>
<td>Projection</td>
<td>the fname of table</td>
<td>[fname] of table</td>
<td>the cuisine of restaurants</td>
</tr>
<tr>
<td>Subquery</td>
<td>the table1 of table2</td>
<td>table1, in_array(id, any(table1 of table2))</td>
<td>reviews of restaurant X</td>
</tr>
<tr>
<td>Join</td>
<td>table1 with their table2</td>
<td>table join table2</td>
<td>restaurants with their reviews</td>
</tr>
<tr>
<td>Aggregate</td>
<td>the number of table</td>
<td>count (table)</td>
<td>The number of restaurants</td>
</tr>
<tr>
<td></td>
<td>the op fname in table</td>
<td>op (fname of table)</td>
<td>The average rating of restaurants</td>
</tr>
<tr>
<td>Aggregate &amp; Group by</td>
<td>the number of table in each fname</td>
<td>count (table by fname)</td>
<td>The number of restaurants</td>
</tr>
<tr>
<td></td>
<td>the op fname1 in table in each fname2</td>
<td>op (fname1 of table by fname2)</td>
<td>The average rating of restaurants</td>
</tr>
<tr>
<td>Ranking</td>
<td>the n table with the min fname</td>
<td>sort (fname asc of table)[1:n]</td>
<td>the 3 restaurants with the min rating</td>
</tr>
<tr>
<td>Quantifier</td>
<td>table1 with table2</td>
<td>table1, contains(table2, any(table2))</td>
<td>restaurants with review with …</td>
</tr>
<tr>
<td></td>
<td>table1 with no table2</td>
<td>table1, !contains(table2, any(table2))</td>
<td>restaurants with no review with …</td>
</tr>
<tr>
<td>Row-wise function</td>
<td>the distance of table from location</td>
<td>[distance(geo, location)] of table</td>
<td>The distance of restaurants from here</td>
</tr>
<tr>
<td></td>
<td>the number of fname in table</td>
<td>[count(fname)] of table</td>
<td>The number of reviews in restaurants</td>
</tr>
</tbody>
</table>
Quiz

• True or false:
  We use the grammar / templates to generate all possible queries and we are done.
Why is NL Hard?

• **Alternatives for just 1 fact:** “*Dr. Smith is Ann’s doctor*”

<table>
<thead>
<tr>
<th>Relation</th>
<th>Part-of-Speech (POS)</th>
<th>Unknown: <em>Ann</em></th>
<th>Unknown: <em>Dr. Smith</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Doctor</td>
<td>Has-a</td>
<td>Who has <em>Dr. Smith</em> as a doctor?</td>
<td>Who does <em>Ann</em> have as a doctor?</td>
</tr>
<tr>
<td></td>
<td>Is-a</td>
<td>Who is <em>Dr. Smith</em> a doctor of?</td>
<td>Who is a doctor of <em>Ann</em>?</td>
</tr>
<tr>
<td></td>
<td>Active Verb</td>
<td>Whom does <em>Dr. Smith</em> treat?</td>
<td>Who treats <em>Ann</em>?</td>
</tr>
<tr>
<td></td>
<td>Passive Verb</td>
<td>Who is treated by <em>Dr. Smith</em>?</td>
<td>By whom is <em>Ann</em> treated?</td>
</tr>
<tr>
<td>Patient</td>
<td>Has-a</td>
<td>Who does <em>Dr. Smith</em> have as a patient?</td>
<td>Who has <em>Ann</em> as a patient?</td>
</tr>
<tr>
<td></td>
<td>Is-a</td>
<td>Who is a patient of <em>Dr. Smith</em>?</td>
<td>Who is <em>Ann</em> a patient of?</td>
</tr>
<tr>
<td></td>
<td>Active Verb</td>
<td>Who <em>consults</em> with <em>Dr. Smith</em>?</td>
<td>With whom does <em>Ann</em> consult?</td>
</tr>
<tr>
<td></td>
<td>Passive Verb</td>
<td>By whom is <em>Dr. Smith</em> consulted?</td>
<td>Who is consulted by <em>Ann</em>?</td>
</tr>
</tbody>
</table>
Property-Level Templates
Based on POS (Part-of-Speech)

alumniOf property in people table

<table>
<thead>
<tr>
<th>POS</th>
<th>Annotation</th>
<th>Example Template</th>
<th>Example utterance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is-a Noun</td>
<td>alumni of &lt;value&gt;</td>
<td><code>table who are [noun phrase] value</code></td>
<td>people who are alumni of Stanford</td>
</tr>
<tr>
<td>Has-a Noun</td>
<td>a &lt;value&gt; degree</td>
<td><code>table with a value [noun phrase]</code></td>
<td>people with a Stanford degree</td>
</tr>
<tr>
<td>Active verb</td>
<td>graduated from &lt;value&gt;</td>
<td><code>table who [verb phrase] value</code></td>
<td>people who graduated from Stanford</td>
</tr>
<tr>
<td>Passive verb</td>
<td>educated at &lt;value&gt;</td>
<td><code>table [passive verb phrase] value</code></td>
<td>people who educated at Stanford</td>
</tr>
<tr>
<td>Adjective</td>
<td>&lt;value&gt;</td>
<td><code>value table</code></td>
<td>Stanford people</td>
</tr>
<tr>
<td>Prepositional</td>
<td>from &lt;value&gt;</td>
<td><code>table [prepositional phrase] value</code></td>
<td>people from Stanford</td>
</tr>
</tbody>
</table>

Note: Not all properties can be referred by these different parts-of-speech.
Outline

1. Are Siri, Alexa, and Google assistant good enough? (Traditional approach)
2. Why do we care about databases?
3. Key Insight: Schema-based approach
4. Query representation: ThingTalk
5. Getting used to natural language as database queries
6. Queries factored into operators and domain knowledge
7. An Overview to Training
Training Data for Semantic Parsing

- What do the developer need to do?
  - Use good field names; declare types
  - Add annotations: different parts of speech for the fields (optional)
  - Supply a sample database
  - Add few-shot annotated examples
- Synthesizer
  - We use altogether 900 templates to generate varieties.
  - Use templates to synthesize a sample of all possible queries with values from database, with perfect annotation
- Paraphraser (manual / automatic)
  - Rephrase statements to make them more natural for the domain

Restaurant

- `id`: `Entity(Restaurant)`
- `geo`: `Location`
  - `address`, `in #`, `near #`, `around #`
- `price`: `Enum(cheap, moderate, expensive)`
  - `# -ly priced`, `#`
- `cuisines`: `Array(Entity(Cuisine))`
  - `# food`, `serves # food`
Quiz

• It is a lot of work to write these 900 templates. Is it worthwhile?
Synthesis + Few Shot + Self-Training

Domain Information

Data Synthesis

Value Datasets

State Machine: Policy Function + Templates

Augmentation

Automatic Paraphrasing

Fine-tuned BART

Few-Shot Annotated Data

2\textsuperscript{nd} Fine-tuned Model
What You Have Learned From This Lecture

• How natural language queries can be mapped to database queries (assuming the right fields exist)
• ThingTalk extends SQL beyond DBs (APIs, events)
  • Tailored more for semantic parsing
• Database: relational operators + domain knowledge in schemas
  • All database queries are combinations of 6 fundamental operations
  • All database operations can be expressed as grammars with formal database operators or natural language constructs
• Training data
  • Can use templates to synthesize a sample to cover the space of queries
  • Synthesized data miss the natural language alternatives
    • User-supplied annotations + paraphrases (automatic/manual)
    • Few-shot per domain
What You Will Learn in the Future

• Concept of synthesize+paraphrase was first introduced by the Overnight system

• Performance was not so good – why?

• The key is to synthesize enough variety – how?

• What does it take to paraphrase automatically?

• What are the experimental results?
Homework 2

• Create a question answering agent for a subset of Wiki
• Goal: to get you familiar with the process
  • An automatic script to get your familiar with the process
  • Write a few annotations and few-shot data to understand the flow
Examples of Projects

• **A Multimodal Building Management Assistant** (Junwen Zheng)
  - Brings digitization to “hardhat” community (with semantic parsing)

• **RUSS: Rapid Universal Support Service**
  (Nancy Xu, Nancy Xu, Sam Masling, Michael Du)
  - Train once and answer calls for different companies
Facility Management MultiModal Assistant
Junwen Zheng, CIFE, Q4 2021, Q1 2022

Construction industry: 2\textsuperscript{nd} least digitized industry
Digital information inaccessible to blue collar workers
• Not tech savvy
• Hands are busy
• On the job, not at the desk
Voice interface: access info on the job naturally
Built with 100 manually annotated commands + synthesized data
for Facilities Management
# Automated Call Agents

<table>
<thead>
<tr>
<th>Create an Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Select the <strong>Account</strong> button in the upper-right corner of Walmart.com</td>
</tr>
<tr>
<td>2. Click <strong>Create an Account</strong></td>
</tr>
<tr>
<td>3. Enter your name and email address on the sign-up screen</td>
</tr>
<tr>
<td>4. Create a password and confirm it</td>
</tr>
<tr>
<td>5. Select <strong>Create Account</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Redeem a Gift Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Find the claim code.</td>
</tr>
<tr>
<td>2. Go to <strong>Redeem a Gift Card</strong>.</td>
</tr>
<tr>
<td>3. Enter your claim code and select <strong>Apply to Your Balance</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Find your AD Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. Click <strong>Ads</strong> to open Ads Manager</td>
</tr>
<tr>
<td>0. Click <strong>Reporting</strong></td>
</tr>
<tr>
<td>0. Click the Ads tab under the graph</td>
</tr>
<tr>
<td>0. Scroll down to the list of ads</td>
</tr>
<tr>
<td>0. The 12-digit number beginning with 687 under the ad name is the ad ID</td>
</tr>
</tbody>
</table>

**Research Question:**

Can we train one universal agent that reads instructions on online help centers?
RUSS
Rapid Universal Support Service
Nancy Xu, Sam Masling, Michael Du, Q4, 2020

• Trained \textit{once} with synthesized data
• Extended ThingTalk for web operations
• Experiment
  • 22 different online help centers
  • 80 tasks, 741 instructions
  • Overall accuracy: 77%
  • Semantic parsing accuracy: 87%
  • Users prefer RUSS over the web