CSC 370

SQL (our first pass)

Intro

- SQL is a very-high-level language
  - We denote "what to do" rather than "how to do it"
  - SQL statements avoid much of the data-manipulation operations needed in C++ or Java
- Ultimately the DBMS determines how best to execute the query
  - Want to balance time and space
  - Query optimization

select-from-where

```
select   <desired attributes>
from     <one or more tables>
where    <condition about tuples in table is true>
```

π <desired attributes> (σ <condition> (<table(s)>))

Our wee little schema

- We will examine SQL queries from the use of our Pubs example
- (Underlined attributes ≡ keys)

```
Beers (name, manf)
Pubs (name, addr, license)
Patrons (name, addr, phone)
Likes (patron, beer)
Sells (bar, beer, price)
Frequents (patron, pub)
```

First example

- Using Beers(name, manf):
  - "What beers are made by Phillips?"

```
select name
from Beers
where manf = 'Phillips';
```

First example: result

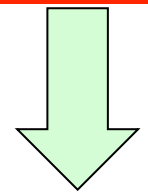
name
Longboat Porter
Amnesiac
Blue Buck
Hop Circle IPA
Dr Funk

- Answer is a relation:
  - Single attribute (name)
  - Name of each beer in the table brewed by Phillips.

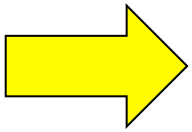
Single-relation query: meaning

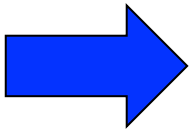
- How do we interpret such a query?
  - Begin with the relation in the **from** clause
  - Apply the selection indicated by the **where** clause
  - Finally, apply the extended projection indicated by the **select** clause
- Usual approach to describing this more precisely:
  - Assuming some theoretical machine that works on relation tuples...
  - ... then list what operations the machine performs on those tuples (i.e., **operational semantics**)

Operational Semantics



name	manf
...	...
...	...
Blue Buck	Phillips
...	...
...	...





Tuple variable t loops over all tuples in relation

Check if t.manf = Phillips

If so, include t.name in the result

Operational semantics

- Think of a **tuple variable** that visits each tuple of the relation in **from** clause
- Check if the tuple currently being visited satisfies the condition in the **where** clause
- If so:
  - Compute the attributes or expressions of the **select** clause...
  - ... and do so by using the components of the tuple we are visiting.

* in select

- Sometimes we simply want all attributes in a relation returned as a query's tuples
  - * means all attributes
- (This is same as leaving off the final π in the equivalent relational-algebra expression.)

```
select *
from beers
where manf = 'Phillips';
```

result

name	manf
Longboat Porter	Phillips
Amnesiac	Phillips
Blue Buck	Phillips
Hop Circle IPA	Phillips
Dr Funk	Phillips

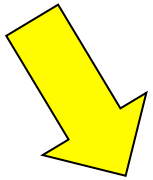
Result has each of the attributes in tuples from Beers that match the **where** clause

Renaming

- Our relational algebra operators include ρ
  - This renames attributes in a schema
- If we need the result to have different attribute names, we can specify this in the **select** clause
  - Use **as <newname>** beside the original attribute name

```
select name as beername, manf
from beers
where manf = 'Phillips';
```

result



beername	manf
Longboat Porter	Phillips
Amnesiac	Phillips
Blue Buck	Phillips
Hop Circle IPA	Phillips
Dr Funk	Phillips

Attribute in the result is **beername** instead of **name**. This does not change the name of the attribute in the original stored relation!

Expression in select clauses

- The extended projection operator in the RA permits expressions to generate columns
- Any expression meaningful with respect to the relation can appear as an element in the select.

```
select pub, beer, price, price*1.15 as pricewithtax
from sells;
```

result

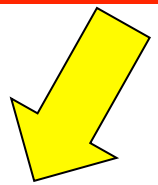
pub	beer	price	pricewithtax
Moe's	Blue	4	4.6
Moe's	Longboat	7	8.05
Cheers	Blue	4.25	4.89
Cheers	Guinness	7.2	8.28
St. Agatha's	Amnesiac	5.5	6.33

Expression in select clauses: constants

- Using Likes(patron, beer)
- We want a constant value to be used for the attribute value in all result tuples

```
select patron,
       'likes Blue' as whoLikesBlue
from likes
where beer = 'Blue';
```

result



patron	whoLikesBlue
Miley	likes Blue
Britney	likes Blue
Justin	likes Blue
Homer	likes Blue

Every tuple will have exactly the same constant value in the **whoLikesBlue** column.

Example: integrating data

- We often aggregate data together using many sources
- This is done in order to answer some query involving the data
- Suppose: Each pub has its own relation Menu(beer, price)
- We can:
  - Create a result relation equivalent to Sells(pub, beer, price)...
  - ... but without having to create a new table.

Integrating data: example

- We assume a relation Menu that is just for Moe's Tavern
- For Moe's we can issue this query:

```
select 'Moe''s', beer, price
from Menu;
```

- ... and the result is a little bit like Sells!
- Later we will see that this kind of result can be used as a subquery for a more complex query.

Complex conditions

- Relational and boolean operators are available in a **where** clause
  - boolean: **and**, **or**, **not**
  - relational: **=, <>, <, >, <=, >=**
  - Other operators produce boolean-value result (i.e., operations on sets)
- Example: Sells(pub, beer, price): Find the price Moe's Tavern charges for Labatt's Blue

```
select price
from Sells
where pub = 'Moe''s' and beer = 'Blue';
```

Patterns

- A condition may compare strings to a pattern
  - <attribute> **like** <pattern>
  - <attribute> **not like** <pattern>
- A <pattern> is a quoted string where:
  - % (percent sign) means any string
  - _ (underscore) means any character
- Example: Using Patrons(name, addr, phone), find those patrons whose exchange is 472

```
select name
from Patrons
where phone like '%472-____';
```

NULL values

- In our discussion of data modeling we often referred to null values
- Tuples in SQL relations can have NULL as a value for one or more attributes
- However, the meaning of NULL can be tricky (i.e., context dependent)
  - NULL as **missing value**: We know Bob's Pub has some address, but we do not know what it is at the time we insert the tuple into the relation.
  - NULL as **inapplicable**: The value of the spouse attribute for an unmarried person is NULL.

Comparing to NULLs

- SQL logic is actually based on three values, not two
  - **true**
  - **false**
  - **unknown**
- Comparing any value (including NULL itself!) with NULL yields **unknown**
- A tuple is in a query answer if and only if the **where** clause is **true** for that tuple
  - That is, if the clause is **false** or **unknown**, the tuple is not in the answer

Understanding three-valued logic

- One way of understanding what unknown means in a boolean expression is instead think of arithmetic
  - **true** = 1.0
  - **false** = 0.0
  - **unknown** = 0.5
- Logical operators:
  - and(m, n) = min(m, n)
  - or(m, n) = max(m, n)
  - not(m) = 1 – m
- Example:
  - **true** and (**false** or not(**unknown**))
  - min(1, max(0, (1 – 0.5)))
  - 0.5

Counter-intuitive!!

```
select pub
from Sells
where price < 2.00 or price >= 2.00;
```

pub	beer	price
Bob's	Blue	NULL

With this instance of Sells, we get an empty result!