CS411
Database Systems

06: SQL

Kazuhiro Minami

SQL = **S**tructured **Q**uery **L**anguage

Standard language for querying and manipulating data

- Has similar capabilities for queries to those in relational algebra
- Support statements for modifying a database (e.g., inserting and deleting tuples) and for declaring a database schema

Many standards: SQL92, SQL2, SQL3, SQL99

- We cover features that conform with SQL99

What is special about SQL?

You describe ***what*** you want,

and the job of the DBMS is to figure out ***how*** to compute what you want efficiently.

(at least in theory)

The basic form of a SQL query is *select-from-where*

SELECT desired attributes

FROM one or more tables

WHERE condition on the rows of the tables

Project out everything not in the final answer

Every table you want to join, together

All the join and selection conditions

Single-Relation Queries

What beers are made by Anheuser-Busch?

Beers(name, manf)

Name	Manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
Super Dry	Asahi

Name
Bud
Bud Lite
Michelob

```
SELECT   name
FROM     Beers
WHERE    manf = 'Anheuser-Busch';
```

In relational algebra: σ [manf = "Anheuser-Busch"] Beers

These simple queries can be translated to relational algebra

1. Begin with the relation in the FROM clause.
2. Apply the selection indicated by the WHERE clause.
3. Apply the projection indicated by the SELECT clause.

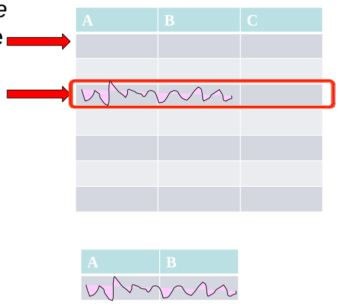
```
SELECT A1, …, An
FROM R
WHERE condition
```

R[*condition*][*A1*, …, *An*]

$\pi_{[A1,\ \ldots,\ An]}\ \sigma_{[condition]}\ R$

Here is a way to think about how the query might be implemented

1. Imagine a *tuple variable* ranging over each tuple of the relation mentioned in FROM.
2. Check if the "current" tuple satisfies the WHERE clause.
3. If so, output the attributes/expressions of the SELECT clause using the components of this tuple.



Put * in the SELECT clause if you don’t want to project out any attributes

Beers(<u>name</u>, manf)

```
SELECT   *
FROM     Beers
WHERE    manf = ‘Anheuser-Busch’;
```

Name	Manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch

Find all US companies whose stock is > $500

Company(sticker, name, country, stockPrice)

```
SELECT   *
FROM     Company
WHERE    country=‘USA’ AND stockPrice > 500
```

Sticker	Name	Country	StockPrice
GOOG	Google	USA	550
GOOG	Apple	USA	485

You can rename the attributes in the result, using “as <new name>”

Beers(<u>name</u>, manf)

```
SELECT   name AS beer, manf
FROM     Beers
WHERE    manf = ‘Anheuser-Busch’;
```

Beer	Manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch

You can use math in the SELECT clause

Sells(<u>bar</u>, <u>beer</u>, price)

Case-insensitive, except inside quoted strings

```
SELECT   bar, bEeR, price*120 AS priceInYen
FROM     Sells;
```

Bar	Beer	PriceInYen
Joe’s	Bud	300
Sue’s	Asahi	360
…	…	…

You can create a new column and give it a constant value, in the SELECT clause

Likes(Drinker, beer)

```
SELECT   drinker,
         'Likes Bud' AS WhoLikesBud
FROM     Likes
WHERE    beer = 'Bud';
```

Drinker	Beer
Sally	Bud
Fred	Bud

Drinker	WhoLikesBud
Sally	Likes Bud
Fred	Likes Bud

Find the price Joe's Bar charges for Bud.

Sells(bar, beer, price)

```
SELECT   price
FROM     Sells
WHERE    bar = 'Joe''s Bar' AND beer = 'Bud';
```

Two single quotes inside a string = one apostrophe

What you can use in the WHERE clause conditions:

constants of any supported type
attribute names of the relation(s) used in the FROM
arithmetic operations: stockprice*2
operations on strings (e.g., "||" for concatenation)

comparison operators: =, <>, <, >, <=, >=
lexicographic order on strings (<)
string pattern matching: s LIKE p
special functions for comparing dates and times

and combinations of the above using AND, OR, NOT, and parentheses

attr LIKE *pattern* does pattern matching on strings

pattern is a quoted string that may contain two special symbols:

Symbol	What It Matches
%	matches any sequence of characters
_	matches any single character

phone **LIKE** '%555-_ _ _ _'
address **LIKE** "%Mountain%"

Find the drinkers with phone prefix 555

Drinkers(name, addr, phone)

```
SELECT   name
FROM     Drinkers
WHERE    phone LIKE '%555-____';
```

Find all US companies whose address contains "Mountain"

Company(sticker, name, address, country, stockPrice)

```
SELECT   *
FROM     Company
WHERE    country="USA" AND
         address LIKE '%Mountain%';
```

What if an attribute value is unknown, or the attribute is inapplicable (e.g., my daughter's spouse)?

Bar	Beer	Price
Jillian's	Bud	2.00
White Horse Inn	Asahi	NULL

```
SELECT   bar
FROM     Sells
WHERE    price < 2.00 OR price >= 2.00;
```

Bar
Jillian's

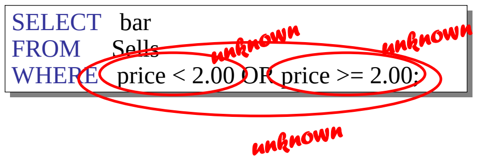
Why???

Conditions involving NULL evaluate to *unknown*, rather than *true* or *false*

Example condition	Evaluates to
'Smith' = 'Smith'	*true*
2 > 6	*false*
'Smith' = NULL	*unknown*
2 < NULL	*unknown*
true AND *unknown*	*unknown*
true OR *unknown*	*true*
false AND *unknown*	*false*
false OR *unknown*	*unknown*
unknown OR *unknown*	*unknown*

A tuple only goes in the answer if its truth value for the WHERE clause is true.

The “law of the excluded middle” doesn’t hold in this 3-valued logic



```
SELECT   bar
FROM     Sells
WHERE    price < 2.00 OR price >= 2.00;
```

Bar	Beer	Price
White Horse Inn	Asahi	NULL

SQL code writers spend a lot of space dealing with NULL values

Can test for NULL explicitly:

x IS NULL

x IS NOT NULL

```
SELECT *
FROM     Person
WHERE  age < 25  OR  age >= 25 OR
age IS NULL
```

The answer includes all Persons!

Exercise 1: online bookstore

Book(<u>isbn</u>, title, publisher, price)
Author(<u>assn</u>, aname, isbn)
Customer(<u>cid</u>, cname, state, city, zipcode)
Buy(<u>tid</u>, cid, isbn, year, month, day)

Q1: Make a list of the ISBNs and titles of books whose price is greater than $1000?

```
SELECT isbn, title
FROM    Book
WHERE price > 1000
```

Multi-Relation Queries

If you need to join several relations, you can list them all in the FROM clause

List the bars that serve a beer that Alice likes.

```
Likes(drinker, beer)     Sells(bar, beer, price)
```

```
SELECT   bar
FROM     Sells, Likes
WHERE    drinker = ‘Alice’ AND
Likes.beer = Sells.beer;
```

This is how we disambiguate attribute names.

π [bar](Sells ⋈ σ [drinker =“Alice”] Likes)

Find the beers liked by at least one person who frequents Murphy’s Pub

```
Likes(drinker, beer)    Frequents(drinker, bar)
```

```
SELECT   beer AS beersWorthKeeping
FROM     Likes, Frequents
WHERE    bar = ‘Murphy’’s Pub’ AND
         Frequents.drinker = Likes.drinker;
```

BeersWorthKeeping
Samuel Adams Pale Ale
…

π [beer] (Likes ⋈ σ [bar = “Murphy’s Pub”] Frequents)

Find names of people living in Champaign who bought snow shovels, and the names of the stores where they bought them

```
Purchase (buyer,  seller,  store,  product)
Person(pname, phoneNumber, city)
```

```
SELECT   pname, store
FROM     Person, Purchase
WHERE    pname = buyer AND city = ‘Champaign’
         AND product = ‘snow shovel’;
```

π [pname, store](σ [city = “Champaign”] Person $\bowtie_{\text{Pname = buyer}}$ σ [product=“snow shovel”] Purchase)

You can also join three or more relations, just like in relational algebra

Find names and phone numbers of people buying telephony products.

```
Product (name,  price, category, maker)
Purchase (buyer,  seller,  store,  product)
Person (name, phoneNumber, city)
```

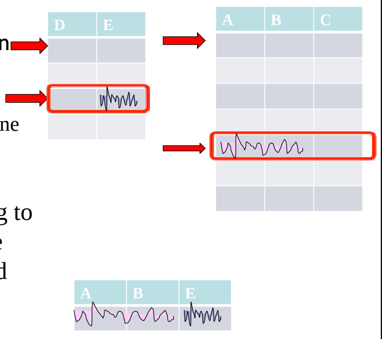
```
SELECT   Person.name, Person.phoneNumber
FROM     Person, Purchase, Product
WHERE    Person.name=Purchase.buyer
    AND  Purchase.product=Product.name
    AND  Product.category=“telephony”
```

What should be in the answer when the query involves a join?

1. Create the cartesian product of all the relations in the FROM clause.
2. Then remove all the tuples that don't satisfy the selection condition in the WHERE clause.
3. Project the remaining tuples onto the list of attributes/expressions in the SELECT clause.

An algorithm for computing the answer

1. Imagine one *tuple variable* for each relation mentioned in FROM. These tuple-variables visit each combination of tuples, one from each relation.
2. Whenever the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause.



Exercise 2: online bookstore

Book(<u>isbn</u>, title, publisher, price)
Author(<u>assn</u>, aname, <u>isbn</u>)
Customer(<u>cid</u>, cname, state, city, zipcode)
Buy(<u>tid</u>, cid, isbn, year, month, day)

Q2: Make a list of the CIDs and customer names who bought books written by 'Barack Obama'?

```
SELECT Customer.cid, Customer.cname
FROM    Author, Buy, Customer
WHERE Customer.cid = Buy.cid AND Buy.isbn = Author.ibn
            AND Author.name = `Barack Obama' ;
```