Principles of Database Systems
CSE 544

Lecture #1
Introduction and SQL
Staff

• Instructor: Dan Suciu
  – CSE 662, suciu@cs.washington.edu
  – Office hour: Wednesdays, 1:30-2:20, CSE 662

• TA:
  – Kevin Kar Wai Lai, kevinlai@cs.washington.edu
  – Office hour: Tuesday, 1:30-2:20, CSE 218
Class Format

• Lectures Tuesday-Thursday, 12-1:30pm

• 4 Homework Assignments

• Reading assignments

• A mini-research project
Announcements

This week is special

• First lecture on Monday
• No lectures on:
  – Tuesday, April 2
  – Thursday, April 4
Textbook and Papers

• Official Textbook:
  – Book available on the Kindle too
  – Use it to read background material
  – You may borrow it, no need to buy

• Other Books
  – Foundations of Databases, by Abiteboul, Hull, Vianu
  – Finite Model Theory, by Libkin
Textbook and Papers

• Nine papers to read and review
  – Mix of old seminal papers and new papers
  – Papers available online on class website
  – Most papers available on Kindle
  – Some papers come from the “red book” [no need to get it]

• Plus a couple of optional readings
Resources

• Web page: http://www.cs.washington.edu/education/courses/cse544/13sp/
  – Lectures
  – Reading assignments
  – Homework assignments
  – Projects

• Mailing list:
  – Announcements, group discussions
Content of the Class

• **Relational Data Model**
  – SQL, Data Models, Relational calculus, Constraints+Views,

• **Systems**
  – Storage, query execution, query optimization, database statistics, parallel databases

• **Theory**
  – Query complexity, query containment, datalog, bounded tree-width

• **Miscellaneous**
  – Transactions, provenance, data privacy
Evaluation

• **Assignments 50%:**
  – Four assignments: programming + theory

• **Project 30%:** Groups of 1-3
  – Small research or engineering. Start thinking now!

• **Paper reviews, class participation 20%:**
  – Individual
  – Due by the evening before the lecture
  – Reading questions are posted on class website
Assignments 50%

- HW1: Data Analysis Pipeline programming
- HW2: Database Systems programming
- HW3: Parallel Data Analytics programming
- HW4: Database Theory theory

We will accept late assignments with valid excuse
Assignments 50%

- **HW1**: Data Analysis Pipeline – posted!
  - Design schema: E/R diagram, tables
  - Install postres, import the DBLP data
  - Transform DBLP data to your schema – SQL
  - Do data analysis – SQL, SQL, SQL, ...
  - Draw graphs – Excel

- **Due**: Monday, April 22, 11:59pm
Project 30%

• Teams: 1-3 students

• Topics: choose one of:
  – A list of mini-research topics (see Website, check updates)
  – Come up with your own (related to your own research)

• Deliverables (see Website for dates)
  – M1: teams April 12
  – M2: project proposal April 26
  – M3: major milestone May 17
  – M4: presentation on Friday June 07, CSE 405
  – M5: final report June 07

• Amount of work may vary widely between groups
Paper Reviews and Class Participation 20%

- **Reviews**: 1/2 page in length
  - Summary of the main points of the paper
  - Critical discussion of the paper

- **Review questions**
  - For some papers, we will post reading questions to help you figure out what to focus on when reading the paper
  - Please address these questions in your reviews

- **Discussions**
  - Ask questions, raise issues, think critically
  - Learn to express your opinion
  - Respect other people’s opinions

- **Grading: credit/no-credit**
  - You can skip one review without penalty
  - MUST submit review BEFORE lecture
  - Individual assignments (but feel free to discuss paper with others)
Goals of the Class

This is a CSE graduate level class!

• Using databases in research:
  – Data analysis pipeline
  – Expert use of database systems (Postgres) and of novel data analysis tools (MapReduce)

• Some (limited) exposure to database internals

• Using database concepts in research:
  – Algorithms/techniques for massive data processing/analysis (sequential and/or parallel)
  – Theory of query complexity, datalog

• Exposure to database research:
  – Query processing, provenance, privacy, theory…
You should have heard about most of:

- E/R diagrams
- Normal forms (1\textsuperscript{st}, 3\textsuperscript{rd})
- SQL
- Relational Algebra
- Indexes, search trees
- Search in a binary tree
- Query optimization (e.g. join reordering)
- Transactions
- PTIME, NP, LOGSPACE
- Logic: $\land$, $\lor$, $\forall$, $\exists$, $\neg$, $\in$
- Reachability in a graph

We will cover these topics in class, but assume some background
Agenda for Today

• Brief overview of a traditional database systems

• SQL
Databases

What is a database?

Give examples of databases
Databases

What is a database?
• A collection of files storing related data

Give examples of databases
• Accounts database; payroll database; UW’s students database; Amazon’s products database; airline reservation database
Database Management System

What is a DBMS?

Give examples of DBMS
**Database Management System**

**What is a DBMS?**

- A big C program written by someone else that allows us to manage efficiently a large database and allows it to persist over long periods of time.

**Give examples of DBMS**

- DB2 (IBM), SQL Server (MS), Oracle, Sybase
- MySQL, Postgres, …
Market Shares

From 2006 Gartner report:

• IBM: 21% market with $3.2BN in sales
• Oracle: 47% market with $7.1BN in sales
• Microsoft: 17% market with $2.6BN in sales
An Example

The Internet Movie Database
http://www.imdb.com

• Entities:
  Actors (1.5M), Movies (1.8M), Directors

• Relationships:
  who played where, who directed what, …
Note

• In other classes at UW (344, 444, 544p):
  – We use IMDB/sqlite and SQL Server for extensive practice of SQL

• In 544:
  – We will use DBLP/postgres, which is more hands-on and more research’y

• If you want to practice more SQL:
  – Let me know and I will arrange for you to have access to the IMDB database and/or to SQL Server.
Tables

**Actor:**

<table>
<thead>
<tr>
<th>id</th>
<th>fName</th>
<th>lName</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>195428</td>
<td>Tom</td>
<td>Hanks</td>
<td>M</td>
</tr>
<tr>
<td>645947</td>
<td>Amy</td>
<td>Hanks</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Casts:**

<table>
<thead>
<tr>
<th>pid</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>195428</td>
<td>337166</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**Movie:**

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>337166</td>
<td>Toy Story</td>
<td>1995</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SELECT * 
FROM Actor

SELECT count(*) 
FROM Actor

SELECT * 
FROM Actor 
LIMIT 50

SELECT * 
FROM Actor 
WHERE lName = 'Hanks'
This query has selections and joins

1.8M actors, 11M casts, 1.5M movies;
How can it be so fast?
How Can We Evaluate the Query?

**Actor:**

<table>
<thead>
<tr>
<th>id</th>
<th>fName</th>
<th>lName</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td>Hanks</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.8M actors

**Casts:**

<table>
<thead>
<tr>
<th>pid</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

11M casts

**Movie:**

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td>1995</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.5M movies

Plan 1: . . . . [ in class ]

Plan 2: . . . . [ in class ]
Evaluating Tom Hanks

Classical query execution
- Index-based selection
- Hash-join
- Merge-join
- Index-join

Classical query optimizations:
- Pushing selections down
- Join reorder

Classical statistics
- Table cardinalities
- # distinct values
- Histograms

\[ \sigma_{\text{Name}='\text{Hanks}'} \]
\[ \sigma_{\text{year}=1995} \]

Actor \quad Casts \quad Movie

Actor \quad Casts \quad Movie
Terminology for Query Workloads

• **OLTP (OnLine-Transaction-Processing)**
  – Many updates: transactions are critical
  – Many “point queries”: access record by key
  – Commercial applications

• **Decision-Support**
  – Many aggregate/group-by queries.
  – Sometimes called *data warehouse*
  – Data analytics
Physical Data Independence

Physical data independence:
• Applications are isolated from changes to the physical organization:
  – Adding or dropping an index
  – (Actor,Movie*)* v.s. (Movie,Actor*)* v.s. (Movie*, Casts*, Actor*)

Translating WHAT to HOW:
• SQL = WHAT we want = declarative
• Relational algebra = HOW to get it = algorithm
• RDBMS are about translating WHAT to HOW
Transactions

- Recovery + Concurrency control
- ACID =
  - Atomicity ( = recovery)
  - Consistency
  - Isolation ( = concurrency control)
  - Durability

- Transactions are critical in business apps, but less important in data analytics and research in general
  - In 544 we discuss them only towards the end
  - In 344, 444, 544p we cover them early and extensively
Client/Server Architecture

- **One server:** stores the database
  - called DBMS or RDBMS
  - Usually a beefed-up system:
    - Can be cluster of servers, or parallel DBMS
    - In 544 you will install the postgres server on your own computer
- **Many clients:** run apps and connect to DBMS
  - Interactive: psql (postgres), Management Studio (SQL Server)
  - Java/C++/C#/… applications
  - Connection protocol: ODBC/JDBC
- Exceptions exists; e.g. SQL Lite
SQL

• Will discuss SQL rather quickly in 1.5 lectures

• Resources for learning SQL:
  – The slides in this lecture and in CSEP544
  – The textbook
  – Postgres: type `h` or `?`

• Start working on HW1!
SQL

• Data Manipulation Language (DML)
  – Querying: SELECT-FROM-WHERE
  – Modifying: INSERT/DELETE/UPDATE

• Data Definition Language (DDL)
  – CREATE/ALTER/DROP
  – Constraints: will discuss these in class
Tables in SQL

<table>
<thead>
<tr>
<th><strong>PName</strong></th>
<th><strong>Price</strong></th>
<th><strong>Category</strong></th>
<th><strong>Manufacturer</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

- **Table name**: Product
- **Attribute names**: PName, Price, Category, Manufacturer
- **Key**: 
- **Tuples or rows**
Creating Tables, Importing Data

CREATE TABLE Product (  
pname varchar(10) primary key,  
price float,  
category char(20),  
manufacturer text  
);

INSERT INTO Product VALUES ('Gizmo', 19.99, 'Gadgets', 'GizmoWorks');  
INSERT INTO Product VALUES ('Powergizmo', 29.99, 'Gadgets', 'GizmoWorks');  
INSERT INTO Product VALUES ('SingleTouch', 149.99, 'Photography', 'Canon');  
INSERT INTO Product VALUES ('MultiTouch', 203.99, 'Household', 'Hitachi');

Better: bulk insert (but database specific!)

COPY Product FROM '/my/directory/datafile.txt'; -- postgres only!
Other Ways to Bulk Insert

CREATE TABLE Product (  
pname varchar(10) primary key,  
price float,  
category char(20),  
manufacturer text  
);

INSERT into Product (  
SELECT ...  
FROM ...  
WHERE...  
);

Quick method: create AND insert

CREATE TABLE Product AS  
SELECT ...  
FROM ...  
WHERE...
Data Types in SQL

• **Atomic types:**
  – Characters: `CHAR(20)`, `VARCHAR(50)`
  – Numbers: `INT`, `BIGINT`, `SMALLINT`, `FLOAT`
  – Others: `MONEY`, `DATETIME`, …
  – Note: an attribute cannot be another table!

• **Record** (aka tuple)
  – Has atomic attributes

• **Table** (relation)
  – A set of tuples

---

No nested tables! (Discussion next…
Normal Forms

• **First Normal Form**
  – All tables must be flat tables
  – Why?

• **Boyce Codd Normal Form**
  – The only functional dependencies are from a key
  – What is a “functional dependency”?
  – Why?

• **Third Normal Form**
  – The only functional dependencies are from keys, except …
    [boring technical condition here]
  – Why?
Normal Forms

• First Normal Form
  – All tables must be flat tables
  – Why? Physical data independence!

• Boyce Codd Normal Form
  – The only functional dependencies are from a key
  – What is a “functional dependency”? 
  – Why? Avoid data anomalies (redundancy, update, delete)

• Third Normal Form
  – The only functional dependencies are from keys, except …
    [boring technical condition here]
  – Why? Because that’s how we can recover all FD’s.

Your schema in HW1 should be in BCNF (easier than it sounds)
Simple Selection Queries in SQL

```
SELECT * FROM Product WHERE category='Gadgets'
```

```
SELECT * FROM Product WHERE category > 'Gadgets'
```

```
SELECT * FROM Product WHERE category LIKE 'Ga%'
```

```
SELECT * FROM Product WHERE category LIKE '%dg%'
```

“selection”
"DISTINCT", "ORDER BY", "LIMIT"

```
SELECT DISTINCT category
FROM Product
```

```
SELECT pname, price, manufacturer
FROM Product
WHERE category='gizmo' AND price > 50
ORDER BY price, pname
LIMIT 20
```
# Keys and Foreign Keys

## Company

<table>
<thead>
<tr>
<th>CName</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

## Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Joins

Product (PName, Price, Category, Manufacturer)
Company (CName, stockPrice, Country)

Find all products under $200 manufactured in Japan;

```
SELECT  x.PName, x.Price  
FROM    Product x, Company y  
WHERE   x.Manufacturer=y.CName
        AND  y.Country='Japan'
        AND  x.Price <= 200
```
Semantics of SQL Queries

```
SELECT a_1, a_2, ..., a_k
FROM R_1 AS x_1, R_2 AS x_2, ..., R_n AS x_n
WHERE Conditions
```

```
Answer = {}
for x_1 in R_1 do
    for x_2 in R_2 do
        .....  
        for x_n in R_n do
            if Conditions
                then Answer = Answer ∪ \{(a_1,\ldots,a_k)\}
return Answer
```
Subqueries

• A *subquery* is another SQL query nested inside a larger query
• Also called *nested queries*
• A subquery may occur in:
  – SELECT
  – FROM
  – WHERE

Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it’s impossible
Universal Quantifiers

Find cities that have a company such that all its products have price < 100

Universal quantifiers are hard! 😞
Universal Quantifiers

Find cities that have a company such that all its products have price < 100

Relational Calculus (a.k.a. First Order Logic) – next lecture

\[ q(y) = \exists x. \text{Company}(x,y) \land (\forall z. \forall p. \text{Product}(z,p,x) \rightarrow p < 100) \]
Universal Quantifiers

De Morgan’s Laws:

\[ \neg (A \land B) = \neg A \lor \neg B \]
\[ \neg (A \lor B) = \neg A \land \neg B \]
\[ \neg \forall x. P(x) = \exists x. \neg P(x) \]
\[ \neg \exists x. P(x) = \forall x. \neg P(x) \]

\[
q(y) = \exists x. \text{Company}(x,y) \land (\forall z. \forall p. \text{Product}(z,p,x) \rightarrow p < 100)
\]

\[
q(y) = \exists x. \text{Company}(x,y) \land \neg (\exists z \exists p. \text{Product}(z,p,x) \land p \geq 100)
\]

\[
\text{theOtherCompanies}(x) = \exists z \exists p. \text{Product}(z,p,x) \land p \geq 100
\]

\[
q(y) = \exists x. \text{Company}(x,y) \land \neg \text{theOtherCompanies}(x)
\]
Universal Quantifiers: \textbf{NOT IN}

\begin{equation*}
\text{theOtherCompanies}(x) = \exists z \exists p. \text{Product}(z, p, x) \land p \geq 100
\end{equation*}

\begin{equation*}
q(y) = \exists x. \text{Company}(x, y) \land \neg \text{theOtherCompanies}(x)
\end{equation*}

\begin{verbatim}
SELECT DISTINCT c.city
FROM Company c
WHERE c.cname \text{NOT IN} (SELECT p.company
FROM Product p
WHERE p.price \geq 100)
\end{verbatim}
Universal Quantifiers: **NOT EXISTS**

theOtherCompanies(x) = \( \exists z \exists p. \text{Product}(z,p,x) \land p \geq 100 \)

q(y) = \( \exists x. \text{Company}(x,y) \land \neg \text{theOtherCompanies}(x) \)

```
SELECT DISTINCT c.city
FROM Company c
WHERE NOT EXISTS (SELECT p.company
                  FROM Product p
                  WHERE c.cname = p.company AND p.price >= 100)
```
Universal Quantifiers: **ALL**

```
SELECT DISTINCT c.city
FROM Company c
WHERE 100 > ALL (SELECT p.price
                  FROM Product p
                  WHERE p.company = c.cname)
```
A Taste of Theory

• Can we unnest the *universal quantifier* query?
  – Can we write it as a simple SELECT-FROM-WHERE query?
Monotone Queries

• A query Q is **monotone** if:
  – Whenever we add tuples to one or more of the tables…
  – … the answer to the query cannot contain fewer tuples

• **Fact**: all unnested queries are monotone
  – Proof: using the “nested for loops” semantics

• **Fact**: A query a universal quantifier is not monotone

• **Consequence**: we cannot unnest a query with a universal quantifier
Queries that must be nested

- Queries with universal quantifiers or with negation
- The drinkers-bars-beers example next
- This is a famous example from textbook on databases by Ullman

**Rule of Thumb:**
Non-monotone queries cannot be unnested. In particular, queries with a universal quantifier cannot be unnested