CSE 510
Web Data Engineering

SQL

cse@buffalo
Applications’ View of a Relational Database Management System (RDBMS)

- Persistent data structure
  - Large volume of data
  - “Independent” from processes using the data
- High-level API for access & modification
  - Automatically optimized
- Transaction management (ACID)
  - Atomicity: all or none happens, despite failures & errors
  - Concurrency
  - Isolation: appearance of “one at a time”
  - Durability: recovery from failures and other errors
Data Structure: Relational Model

• **Relational Databases:** Schema + Data

• **Schema:**
  - collection of *tables* (also called *relations*)
  - each table has a set of *attributes*
  - no repeating relation names, no repeating attributes in one table

• **Data** (also called *instance*):
  - set of *tuples*
  - tuples have one *value* for each attribute of the table they belong

---

<table>
<thead>
<tr>
<th><strong>Movie</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
</tr>
<tr>
<td>Wild</td>
</tr>
<tr>
<td>Sky</td>
</tr>
<tr>
<td>Reds</td>
</tr>
<tr>
<td>Tango</td>
</tr>
<tr>
<td>Tango</td>
</tr>
<tr>
<td>Tango</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Schedule</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Theater</strong></td>
</tr>
<tr>
<td>Odeon</td>
</tr>
<tr>
<td>Forum</td>
</tr>
<tr>
<td>Forum</td>
</tr>
</tbody>
</table>
Example Problem:

- Represent the students and Fall classes of the CSE department, including the list of students who take each class.
- Students have UB ID, first name and last name.
- Classes have a name, a number, date code (TR, MW, MWF) and start/end time.
- A student enrolls for a number of credits in a class.

Solution:...
Programming Interface: JDBC/ODBC

- How client opens connection with a server
- How access & modification commands are issued
- ...

UB CSE 510 Web Data Engineering
Access (Query) & Modification Language: SQL

• SQL
  – used by the database user
  – *declarative*: we only describe *what* we want to retrieve
  – based on tuple relational calculus

• The result of a query is always a table (regardless of the query language used)

• Internal Equivalent of SQL: Relational Algebra
  – used internally by the database system
  – *procedural* (operational): we describe *how* we retrieve

• CSE462, CSE562
SQL Queries: The Basic From

- Basic form
  \[
  \text{SELECT } A_1, \ldots, A_N \\
  \text{FROM } R_1, \ldots, R_M \\
  \text{WHERE } <\text{condition}> 
  \]

- \text{WHERE} clause is optional
- When more than one relations in the \text{FROM} clause have an attribute named \(A\), we refer to a specific \(A\) attribute as \(<\text{RelationName}>.A\)

Find names of all students
Find all students whose first name is John
Find the students registered for CSE510
SQL Queries: Aliases

- Use the same relation more than once in the `FROM` clause
- Tuple variables
- **Problem:** Find the classes taken by students who take CSE510
SQL Queries: Nesting

- The *WHERE* clause can contain predicates of the form
  - `attr/value IN <query>`
  - `attr/value NOT IN <query>`

- The predicate is satisfied if the `attr` or `value` appears in the result of the nested `<query>`

- Also
  - `EXISTS <query>`
  - `NOT EXISTS <query>`

Find the CSE510 students who take a TR 5:00pm class
Universal Quantification by Negation

Problem:
• Find the students that take every class “John Smith” takes

Rephrase:
• Find the students such that there is no class that “John Smith” takes and they do not take
SQL Queries: Aggregation & Grouping

- Aggregate functions: `SUM`, `AVG`, `COUNT`, `MIN`, `MAX`, and recently user defined functions as well
- `GROUP BY`

**Example**: Find the average salary of all employees:

```
SELECT AVG(Salary) AS AvgSal
FROM Employee
```

<table>
<thead>
<tr>
<th>AvgSal</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.5</td>
</tr>
</tbody>
</table>

**Example**: Find the average salary for each department:

```
SELECT Dept, AVG(Salary) AS AvgSal
FROM Employee
GROUP BY Dept
```

<table>
<thead>
<tr>
<th>Dept</th>
<th>AvgSal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toys</td>
<td>40</td>
</tr>
<tr>
<td>PCs</td>
<td>45</td>
</tr>
</tbody>
</table>
SQL Grouping:
Conditions that Apply on Groups

• **HAVING** `<condition>` may follow a **GROUP BY** clause
• If so, the condition applies to each group, and groups not satisfying the condition are eliminated

• **Example**: Find the average salary in each department that has more than 1 employee:

  ```sql
  SELECT Dept, AVG(Salary) AS AvgSal
  FROM Employee
  GROUP BY Dept
  HAVING COUNT(Name) > 1
  ```
Aggregation Can Involve Many Tables

- **Problem:** List students and the number of credits for which they have registered
SQL: More Bells and Whistles ...

- Select all attributes using *
- Pattern matching conditions
  - `<attr>` LIKE `<pattern>`

Retrieve all student attributes of currently enrolled students

Retrieve all students whose name contains “Ta”

```
SELECT *
FROM Students
WHERE name LIKE "%Ta%"
```
...and a Few “Dirty” Points

- **Duplicate elimination** must be explicitly requested
  
  ```sql
  SELECT DISTINCT ...
  FROM ...
  WHERE ...
  ```

- **Null values**
  - All comparisons involving NULL are **false** by definition
  - All aggregation operations, except `COUNT(*)`, ignore NULL values
Null Values and Aggregates

• Example:

```
SELECT COUNT(a), COUNT(b), AVG(b), COUNT(*)
FROM R
GROUP BY a
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>count(a)</th>
<th>count(b)</th>
<th>avg(b)</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>null</td>
<td>2</td>
</tr>
</tbody>
</table>
SQL as a Data Manipulation Language: Insertions

- Inserting tuples
  \[
  \text{INSERT INTO } R(A_1, \ldots, A_k) \\
  \text{VALUES } (v_1, \ldots, v_k);
  \]
- Some values may be left NULL
- Use results of queries for insertion
  \[
  \text{INSERT INTO } R \\
  \text{SELECT } \ldots \\
  \text{FROM } \ldots \\
  \text{WHERE } \ldots
  \]

- Insert in Students “John Doe” with UB ID 88888888
- Insert all CSE510 students into CSE636
SQL as a Data Manipulation Language: Updates and Deletions

- Deletion basic form: delete every tuple that satisfies \(<\text{cond}>\):
  
  \[
  \text{DELETE FROM}\ R \\
  \text{WHERE}\ \langle\text{cond}\rangle
  \]

- Update basic form: update every tuple that satisfies \(<\text{cond}>\) in the way specified by the \text{SET} clause:
  
  \[
  \text{UPDATE}\ R \\
  \text{SET}\ \text{A}_1=\langle\text{exp}_1\rangle,\ldots,\text{A}_k=\langle\text{exp}_k\rangle \\
  \text{WHERE}\ \langle\text{cond}\rangle
  \]

- Delete “John Doe”

- Update the registered credits of all CSE510 students to 4