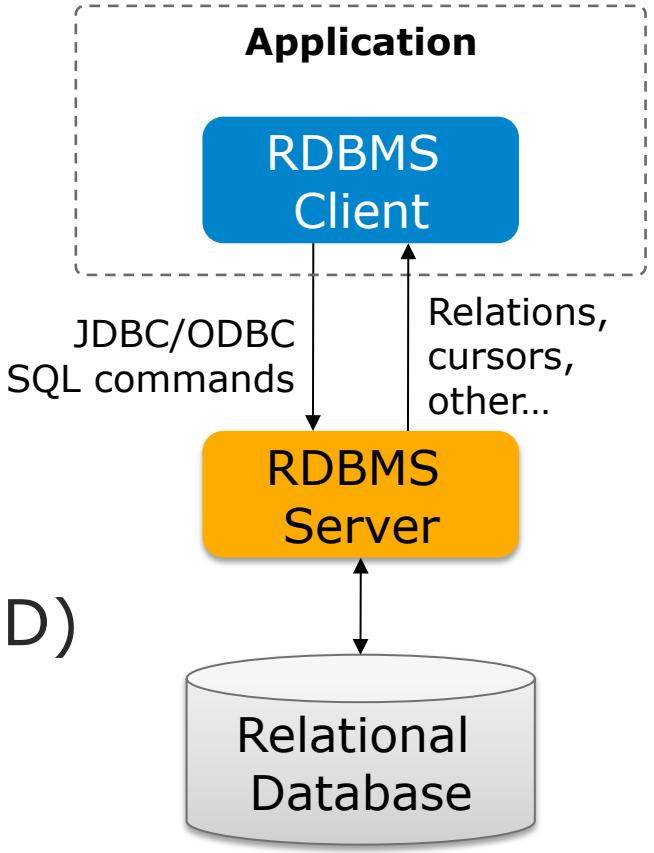
CSE 510
Web Data Engineering

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

Movie

Title	Director	Actor
Wild	Lynch	Winger
Sky	Berto	Winger
Reds	Beatty	Beatty
Tango	Berto	Brando
Tango	Berto	Winger
Tango	Berto	Snyder

Schedule

Theater	Title
Odeon	Wild
Forum	Reds
Forum	Sky

Data Structure: Relational Model

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
- …

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

```
SELECT A1,…,AN
FROM R1,…,RM
WHERE <condition>
```

- **WHERE** clause is optional
- When more than one relations in the **FROM** clause have an attribute named **A**, we refer to a specific **A** attribute as **<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**

Employee

Name	Dept	Salary
Joe	Toys	45
Nick	PCs	50
Jim	Toys	35
Jack	PCs	40

Example: Find the average salary of all employees:

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

AvgSal
42.5

Example: Find the average salary for each department:

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

Dept	AvgSal
Toys	40
PCs	45

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:

```
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

```
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:

R

a	b
x	1
x	2
x	null
null	null
null	null

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

count(a)	count(b)	avg(b)	count(*)
3	2	1.5	3
0	0	null	2

SQL as a Data Manipulation Language: Insertions

- Inserting tuples

```
INSERT INTO R(A1,…,Ak)
    VALUES (v1,…,vk);
```

- Some values may be left NULL
- Use results of queries for insertion

```
INSERT INTO R
    SELECT …
    FROM …
    WHERE …
```

- 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 **<cond>**:

```
DELETE FROM R
WHERE <cond>
```

- Update basic form: update every tuple that satisfies **<cond>** in the way specified by the **SET** clause:

```
UPDATE R
SET A1=<exp1>,…,Ak=<expk>
WHERE <cond>
```

- Delete "John Doe"

- Update the registered credits of all CSE510 students to 4