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

- Applications: ...........
- Persistent data structure
  - Large volume of data
  - “Independent” from processes using the data
- High-level programming interface 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 (also called scheme):
    - 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

Example Problem:
Represent the students and Winter classes of the CSE department, including the list of students who take each class.
Students have UCSD ID, first name and last name.
Classes have a name, a number, date code (TuTh, MW, MWF) and hour start/end.
A student enrolls for a number of units in a class.
Solution:....

Programming Interface: JDBC/ODBC

- How client opens connection with 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

- 132A, 132B

SQL Queries: The Basic From

- Basic form
  
  ```sql
  SELECT DISTINCT a1, …, aN
  FROM R1, …, RM
  WHERE condition
  ```

- WHERE clause is optional
- When more than one relations of the FROM 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 who take “CSE 135”

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 135
SQL Queries: Nesting

- The WHERE clause can contain predicates of the form
  - attr/value IN <SQL query>
  - attr/value NOT IN <SQL query>
- The predicate is satisfied if the attr or value appears in the result of the nested <SQL query>
- Also
  - EXISTS <SQL query>
  - NOT EXISTS <SQL query>
- If the subquery result is not empty

Find the CSE135 students who take a TuTh 5:00pm class

Universal quantification by negation

Find the students that every class that "John Smith" takes they also take

Find the students such that there is no class that John Smith takes and they do not take

SQL Queries: Aggregation and Grouping

- Aggregate functions: AVG, COUNT, MIN, MAX, SUM, and recently user defined functions as well
- Group-by

Find the average salary of all employees

```
SELECT AvgSal=Avg(Salary)
FROM Employee
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Toys</td>
<td>45</td>
</tr>
<tr>
<td>Nick</td>
<td>PCs</td>
<td>50</td>
</tr>
<tr>
<td>Jim</td>
<td>Toys</td>
<td>35</td>
</tr>
<tr>
<td>Jack</td>
<td>PCs</td>
<td>40</td>
</tr>
</tbody>
</table>

Find the average salary for each department

```
SELECT Dept, AvgSal=Avg(Salary)
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** clause

Find the average salary of for each department that has more than 1 employee

SELECT Dept, AvgSal=(Avg(Salary))
FROM Employee
GROUP-BY Dept
HAVING COUNT(Name)>1

Aggregation can also involve many tables

List students and the number of units they have enrolled

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 Title 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

### SQL as a Data Manipulation Language: Insertions

- **Inserting tuples**
  - `INSERT INTO R(A₁,...,Aᵦ) VALUES (v₁,...,vᵦ);`
- some values may be left NULL
- use results of queries for insertion
  - `INSERT INTO R... <query>`

**Insert in Students “John Doe”; his UCSD id is 135791**

**Insert all CSE135 students into CSE132A**

### 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>, ..., Aᵦ=<expᵦ> WHERE <cond>`

**Delete “John Doe”**

**Update the enrolled units of all CSE135 students to 4**
How to design a database and avoid bad designs

• With experience...
• Learn in 132A normalization rules of database design
• Think **entities and relationships** – translate to relations

---

**E/R-based design**

---

**Basic translation from E/R to relational schema**

• For every entity create corresponding table
  – Include an ID attribute even if not in E/R
• For every relationship create table
  – For each referenced entity \( E_i \) include foreign key attribute referencing ID of \( E_i \)
A studio has contracted with a particular star to act in a particular movie.
“Subclassing”

Transaction Management

- Transaction: Collection of actions that maintain the consistency of the database if ran to completion & isolated
- Goal: Guarantee integrity and consistency of data despite
  - Concurrency
  - Failures
- Concurrency Control
- Recovery

Example Concurrency & Failure Problems

- Consider the "John & Mary" checking & savings account
  - C: checking account balance
  - S: savings’ account balance
- Check-to-Savings transfer transaction moves $X from C to S
  - If it runs in the system alone and to completion the total sum of C and S stays the same

```c2s(X=100)
Read(C);
C:=C-100
Write(C)
Read(S)
S:=S+100
Write(S)
```
Example Failure Problem & Recovery Module’s Goal

- Database is in inconsistent state after machine restarts
- It is not the developer’s problem to account for crashes
- Recovery module guarantees that all or none of transaction happens and its effects become “durable”

```
C2S(X=100)
Read(C);
C:=C-100
Write(C)
CPU HALTS
Read(S)
S:=S+100
Write(S)
```

Example Concurrency Problem & Concurrency Control Module’s Goals

- If multiple transactions run in sequence the resulting database is consistent
- Serial schedules – De facto correct

```
Serial Schedule
Read(C);
C:=C+100
Write(C)
Read(S)
S:=S-100
Write(S)
Read(C)
C:=C+50
Write(C)
Read(S)
S:=S-50
Write(S)
```

Example Concurrency Problem & Concurrency Control Module’s Goals

- Databases allow transactions to run in parallel

```
Good Schedule w/ Concurrency
Read(C);
C:=C+100
Write(C)
Read(C)
C:=C+50
Write(C)
Read(S)
S:=S-100
Write(S)
Read(S)
S:=S-50
Write(S)
```

```
Example Concurrency Problem & Concurrency Control Module’s Goals

- “Bad” interleaved schedules may leave database in inconsistent state
- Developer should not have to account for parallelism
- Concurrency control module guarantees serializability
  - only schedules equivalent to serial ones happen

<table>
<thead>
<tr>
<th>Bad Schedule w/ Concurrency</th>
<th>Read(C)</th>
<th>C:=C+100</th>
<th>Read(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write(C)</td>
<td>C:=C+50</td>
<td>Write(C)</td>
<td>Read(S)</td>
</tr>
<tr>
<td></td>
<td>S:=S-50</td>
<td>Write(S)</td>
<td>Read(S)</td>
</tr>
<tr>
<td></td>
<td>S:=S-100</td>
<td>Write(S)</td>
<td></td>
</tr>
</tbody>
</table>