EECS 647: Introduction to Database Systems

Instructor: Luke Huan

Spring 2009
Stating Points

- A database
- A database management system
- A miniworld
- A data model
  - Conceptual model
  - Relational model
So What? Database Schemas

- **A database schema** is a description of a database, using a given data model (relational model by default).
- **External schemas** (user views) describe how users see the data.
- **Conceptual schema** defines logical structure
- **Internal schema** describes the physical storage structure of the database.
Data Independence

- Applications insulated from how data is structured and stored.
- **Logical data independence**: the capacity to change the conceptual schema without having to change the external schema
- **Physical data independence**: the capacity to change the internal schema without having to change the conceptual schema
- *Q: Why is this particularly important for DBMS?*
O.K. So, how to model a miniworld?

- Using conceptual model such as ER model
Where does a conceptual model lead us to?

- A tabular view
Now you have a draft, how do you improve your design?

- Using integrity constraints
  - Attributes value must come from its domain
  - Every relation must have a primary key
  - The primary key value in a tuple can not be NULL
  - The foreign key value in a referenced tuple must exist or the foreign key value in the referencing tuple is NULL
How do you improve relational data base design (cont.)?

- Using functional dependency
  - Non-key FD always leads to redundancy
  - BCNF normal form
    - Pick up a non-key FD, do binary decomposition
- First normal form: no attribute may be composite or multi-valued
- (2\textsuperscript{nd} normal form and 3\textsuperscript{rd} normal form are coming… )
What could we do about relations?

- Relational algebra expression
  - Using temporary variable
  - Identifying information source
  - Pay attention to non monotonic operation
Review

- **SELECT** a list of attributes
- **FROM** a list of relations
- **WHERE** condition;

- Condition may have logical operators AND, OR, NOT
- Condition may have comparison operators: <, <=, <>, >=, >
- String comparison may use “=” (exactly match) or “LIKE” (matching with regular expressions)
  - %, _, \n- (Arithmetic) expressions of attributes are allowed
# Examples of bag operations

<table>
<thead>
<tr>
<th>Bag1</th>
<th>Bag2</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>fruit</td>
</tr>
<tr>
<td>Apple</td>
<td>Apple</td>
</tr>
<tr>
<td>Apple</td>
<td>Orange</td>
</tr>
<tr>
<td>Orange</td>
<td>Orange</td>
</tr>
</tbody>
</table>

- **Bag1 UNION ALL Bag2**
  - fruit
  - Apple
  - Apple
  - Apple
  - Orange
  - Orange

- **Bag1 INTERSECT ALL Bag2**
  - fruit
  - Apple
  - Orange

- **Bag1 EXCEPT ALL Bag2**
  - fruit
  - Apple
  - Orange
Exercise

- SELECT sid, 2009 - age,
  FROM STUDENT
  WHERE name LIKE '%John%' OR GPA > 3.6;

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>19</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
<td>18</td>
<td>2.9</td>
</tr>
<tr>
<td></td>
<td>Age</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>1988</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1123</td>
<td>2009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>2222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1204</td>
<td>2222</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1306</td>
<td>188</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Aggregates

- Standard SQL aggregate functions: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`

- Example: number of students under 18, and their average GPA
  - `SELECT COUNT(*), AVG(GPA) FROM Student WHERE age < 18;`
  - `COUNT(*)` counts the number of rows
Aggregates with DISTINCT

- Example: How many students are taking classes?
  - SELECT COUNT (SID)
    FROM Enroll;
  - SELECT COUNT(DISTINCT SID)
    FROM Enroll;
GROUP BY

- SELECT ... FROM ... WHERE ...
  GROUP BY list_of_columns;

Example: find the average GPA for each age group
- SELECT age, AVG(GPA)
  FROM Student
  GROUP BY age;
Operational semantics of GROUP BY

SELECT ... FROM ... WHERE ... GROUP BY ...;

- Compute FROM ($\times$)
- Compute WHERE ($\sigma$)
- Compute GROUP BY: group rows according to the values of GROUP BY columns
- Compute SELECT for each group ($\pi$)
  - For aggregation functions with DISTINCT inputs, first eliminate duplicates within the group
  - Number of groups = number of rows in the final output
### Example of computing GROUP BY

```
SELECT age, AVG(GPA) FROM Student GROUP BY age;
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>19</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
<td>19</td>
<td>2.9</td>
</tr>
</tbody>
</table>

#### Compute GROUP BY: group rows according to the values of GROUP BY columns

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>19</td>
<td>3.8</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
<td>19</td>
<td>2.9</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
<td>22</td>
<td>3.4</td>
</tr>
</tbody>
</table>

#### Compute SELECT for each group

<table>
<thead>
<tr>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>19</td>
<td>3.35</td>
</tr>
<tr>
<td>22</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Aggregates with no GROUP BY

- An aggregate query with no `GROUP BY` clause represents a special case where all rows go into one group.

```
SELECT AVG(GPA) FROM Student;
```

Compute aggregate over the group:

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>19</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
<td>19</td>
<td>2.9</td>
</tr>
</tbody>
</table>

Group all rows into one group:

```
gpa
3.24
```

2/24/2009  Luke Huan  Univ. of Kansas
Restriction on SELECT

- If a query uses aggregation/group by, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column
- This restriction ensures that any SELECT expression produces only one value for each group
Examples of invalid queries

- SELECT SID, age FROM Student GROUP BY age;
  - Recall there is one output row per group
  - There can be multiple SID values per group

- SELECT SID, MAX(GPA) FROM Student;
  - Recall there is only one group for an aggregate query with no GROUP BY clause
  - There can be multiple SID values
  - Wishful thinking (that the output SID value is the one associated with the highest GPA) does NOT work
HAVING

- Used to filter groups based on the group properties (e.g., aggregate values, GROUP BY column values)

- SELECT ... FROM ... WHERE ... GROUP BY ... HAVING condition;
  - Compute FROM (×)
  - Compute WHERE (σ)
  - Compute GROUP BY: group rows according to the values of GROUP BY columns
  - Compute HAVING (another σ over the groups)
  - Compute SELECT (π) for each group that passes HAVING
HAVING examples

- Find the average GPA for each age group over 10
  - `SELECT age, AVG(GPA)
    FROM Student
    GROUP BY age
    HAVING age > 10;`
  - Can be written using `WHERE` without table expressions

- List the average GPA for each age group with more than a hundred students
  - `SELECT age, AVG(GPA)
    FROM Student
    GROUP BY age
    HAVING COUNT(*) > 100;`
  - Can be written using `WHERE` and table expressions
A First Touch of Subqueries

- Use query result as a table
  - In set and bag operations, \textit{FROM} clauses, etc.
  - A way to “nest” queries
- Example: names of students who are in more clubs than classes

\begin{verbatim}
SELECT DISTINCT name 
FROM Student,
  (SELECT SID FROM ClubMember) 
EXCEPT ALL
  (SELECT SID FROM Enroll) ) AS S 
WHERE Student.SID = S.SID;
\end{verbatim}