DATABASE DESIGN

CS2141 - Software Development using C/C++
Goals of Design

* Don’t require on-the-fly modification of structure
* Make database structure easy to understand
* All records should be unique & unambiguous
* Allow the widest range of useful queries possible
* Eliminate repeated fields
* Minimize redundant storage of data
* Only one sort of record per table
No On-The-Fly Modification

* Changing database structure might break existing queries
* All existing records must be updated
  * Potentially huge performance hit during update
  * Might destroy your data
* NOTE: If you’re still developing the database, changes are fine
Easy To Understand Structure

- Someone will have to maintain the database eventually
- Provide useful column names
- Avoid weird acronyms and abbreviations when possible
- Avoid using spaces and escaped characters in names
- Pick a consistent naming standard and stick with it
## A Simple Table

<table>
<thead>
<tr>
<th>Student</th>
<th>Activity 1</th>
<th>Cost 1</th>
<th>Activity 2</th>
<th>Cost 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>Tennis</td>
<td>$36</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>Jane Bloggs</td>
<td>Squash</td>
<td>$40</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>John Smith</td>
<td>Tennis</td>
<td>$36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mark Antony</td>
<td>Swimming</td>
<td>$15</td>
<td>Golf</td>
<td>$47</td>
</tr>
</tbody>
</table>
## Prevent Ambiguity

<table>
<thead>
<tr>
<th>StudentID</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>John Smith</td>
</tr>
<tr>
<td>100</td>
<td>Jane Bloggs</td>
</tr>
<tr>
<td>182</td>
<td>John Smith</td>
</tr>
<tr>
<td>219</td>
<td>Mark Antony</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StudentID</th>
<th>Activity 1</th>
<th>Cost 1</th>
<th>Activity 2</th>
<th>Cost 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>Tennis</td>
<td>$36</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>100</td>
<td>Squash</td>
<td>$40</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>182</td>
<td>Tennis</td>
<td>$36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>Swimming</td>
<td>$15</td>
<td>Golf</td>
<td>$47</td>
</tr>
</tbody>
</table>
Maximizing Useful Queries

<table>
<thead>
<tr>
<th>Students</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudentID</td>
<td>FirstName</td>
</tr>
<tr>
<td>84</td>
<td>John</td>
</tr>
<tr>
<td>100</td>
<td>Jane</td>
</tr>
<tr>
<td>182</td>
<td>John</td>
</tr>
<tr>
<td>219</td>
<td>Mark</td>
</tr>
</tbody>
</table>
# Eliminate Repeated Fields

## Students

<table>
<thead>
<tr>
<th>StudentID</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>100</td>
<td>Jane</td>
<td>Bloggs</td>
</tr>
<tr>
<td>182</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>219</td>
<td>Mark</td>
<td>Antony</td>
</tr>
</tbody>
</table>

## Activities

<table>
<thead>
<tr>
<th>StudentID</th>
<th>Activity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>Tennis</td>
<td>$36</td>
</tr>
<tr>
<td>84</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>100</td>
<td>Squash</td>
<td>$40</td>
</tr>
<tr>
<td>100</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>182</td>
<td>Tennis</td>
<td>$36</td>
</tr>
<tr>
<td>219</td>
<td>Swimming</td>
<td>$15</td>
</tr>
<tr>
<td>219</td>
<td>Golf</td>
<td>$47</td>
</tr>
</tbody>
</table>
Minimize Redundant Storage

<table>
<thead>
<tr>
<th>StudentID</th>
<th>FirstName</th>
<th>LastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>100</td>
<td>Jane</td>
<td>Bloggs</td>
</tr>
<tr>
<td>182</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>219</td>
<td>Mark</td>
<td>Antony</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StudentID</th>
<th>ActivityID</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>3</td>
</tr>
<tr>
<td>84</td>
<td>15</td>
</tr>
<tr>
<td>100</td>
<td>18</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>182</td>
<td>3</td>
</tr>
<tr>
<td>219</td>
<td>15</td>
</tr>
<tr>
<td>219</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ActivityID</th>
<th>Activity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Tennis</td>
<td>$36</td>
</tr>
<tr>
<td>15</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>18</td>
<td>Squash</td>
<td>$40</td>
</tr>
<tr>
<td>22</td>
<td>Golf</td>
<td>$47</td>
</tr>
</tbody>
</table>
Data Types

* Each column has an associated data type
* Most database engines enforce types
* SQLite supports five types (but does not enforce them):
  * NULL, INTEGER, REAL, TEXT, BLOB
* Dates/times may be stored as text, reals, or integers
Primary Keys

- Mechanism to guarantee a record is uniquely identifiable
- One primary key per table
- Could be any type, but integers are common
- Enforces uniqueness of key on insert
- INTEGER PRIMARY KEY will supply a value if not specified
SQL Syntax

* CREATE TABLE [IF NOT EXISTS] table_name
  (column_name type[, column_name type]);

* eg:
  CREATE TABLE Students
  (StudentID INTEGER PRIMARY KEY,
   FirstName TEXT,
   LastName TEXT);