Muddiest Points

- Connecting a design to an implementation
- Normalization
- Many-to-many
- Operators
Databases

• Database
  – Collection of data, organized to support access
  – Models some aspects of reality

• DataBase Management System (DBMS)
  – Software to create and access databases

• Relational Algebra
  – Special-purpose programming language
Structured Information

• **Field**  
  An “atomic” unit of data
  – number, string, true/false, …

• **Record**  
  A collection of related fields

• **Table**  
  A collection of related records
  – Each record is one row in the table
  – Each field is one column in the table

• **Primary Key**  
  The field that identifies a record
  – Values of a primary key must be unique

• **Database**  
  A collection of tables
A Simple Example

primary key
Registrar Example

- Which students are in which courses?

- What do we need to know about the students?
  - first name, last name, email, department

- What do we need to know about the courses?
  - course ID, description, enrolled students, grades
## A “Flat File” Solution

### Discussion Topic

**Why is this a bad approach?**

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Department ID</th>
<th>Department</th>
<th>Course ID</th>
<th>Course description</th>
<th>Grades</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arrows</td>
<td>John</td>
<td>EE</td>
<td>EE</td>
<td>lbsc690</td>
<td>Information Technology</td>
<td>90</td>
<td>jarrows@wam</td>
</tr>
<tr>
<td>1</td>
<td>Arrows</td>
<td>John</td>
<td>EE</td>
<td>Elec Engin</td>
<td>ee750</td>
<td>Communication</td>
<td>95</td>
<td>ja_2002@yahoo</td>
</tr>
<tr>
<td>2</td>
<td>Peters</td>
<td>Kathy</td>
<td>HIST</td>
<td>HIST</td>
<td>lbsc690</td>
<td>Information Technology</td>
<td>95</td>
<td>kpeters2@wam</td>
</tr>
<tr>
<td>2</td>
<td>Peters</td>
<td>Kathy</td>
<td>HIST</td>
<td>history</td>
<td>hist405</td>
<td>American History</td>
<td>80</td>
<td>kpeters2@wma</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Chris</td>
<td>HIST</td>
<td>history</td>
<td>hist405</td>
<td>American History</td>
<td>90</td>
<td>smith2002@glue</td>
</tr>
<tr>
<td>4</td>
<td>Smith</td>
<td>John</td>
<td>CLIS</td>
<td>Info Sci</td>
<td>lbsc690</td>
<td>Information Technology</td>
<td>98</td>
<td>js03@wam</td>
</tr>
</tbody>
</table>
Goals of “Normalization”

• Save space
  – Save each fact only once

• More rapid updates
  – Every fact only needs to be updated once

• More rapid search
  – Finding something once is good enough

• Avoid inconsistency
  – Changing data once changes it everywhere
Relational Algebra

- Tables represent “relations”
  - Course, course description
  - Name, email address, department

- Named fields represent “attributes”

- Each row in the table is called a “tuple”
  - The order of the rows is not important

- Queries specify desired conditions
  - The DBMS then finds data that satisfies them
# A Normalized Relational Database

## Student Table

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Department ID</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arrows</td>
<td>John</td>
<td>EE</td>
<td>jarrows@wam</td>
</tr>
<tr>
<td>2</td>
<td>Peters</td>
<td>Kathy</td>
<td>HIST</td>
<td>kpeters2@wam</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Chris</td>
<td>HIST</td>
<td>smith2002@glue</td>
</tr>
<tr>
<td>4</td>
<td>Smith</td>
<td>John</td>
<td>CLIS</td>
<td>js03@wam</td>
</tr>
</tbody>
</table>

## Department Table

<table>
<thead>
<tr>
<th>Department ID</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE</td>
<td>Electronic Engineering</td>
</tr>
<tr>
<td>HIST</td>
<td>History</td>
</tr>
<tr>
<td>CLIS</td>
<td>Information Studies</td>
</tr>
</tbody>
</table>

## Course Table

<table>
<thead>
<tr>
<th>Course ID</th>
<th>Course Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbsc690</td>
<td>Information Technology</td>
</tr>
<tr>
<td>ee750</td>
<td>Communication</td>
</tr>
<tr>
<td>hist405</td>
<td>American History</td>
</tr>
</tbody>
</table>

## Enrollment Table

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Course ID</th>
<th>Grades</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lbsc690</td>
<td>90</td>
</tr>
<tr>
<td>1</td>
<td>ee750</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>lbsc690</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>hist405</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>hist405</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>lbsc690</td>
<td>98</td>
</tr>
</tbody>
</table>
Approaches to Normalization

• For simple problems (like the homework)
  – Start with “binary relationships”
    • Pairs of fields that are related
  – Group together wherever possible
  – Add keys where necessary

• For more complicated problems
  – Entity relationship modeling (LBSC 670)
Example of Join

### Student Table

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Department ID</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arrows</td>
<td>John</td>
<td>EE</td>
<td>jarrows@wam</td>
</tr>
<tr>
<td>2</td>
<td>Peters</td>
<td>Kathy</td>
<td>HIST</td>
<td>kpeters2@wam</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Chris</td>
<td>HIST</td>
<td>smith2002@glue</td>
</tr>
<tr>
<td>4</td>
<td>Smith</td>
<td>John</td>
<td>CLIS</td>
<td>js03@wam</td>
</tr>
</tbody>
</table>

### Department Table

<table>
<thead>
<tr>
<th>Department ID</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE</td>
<td>Electronic Engineering</td>
</tr>
<tr>
<td>HIST</td>
<td>History</td>
</tr>
<tr>
<td>CLIS</td>
<td>Information Studies</td>
</tr>
</tbody>
</table>

### “Joined” Table

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Department ID</th>
<th>Department</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arrows</td>
<td>John</td>
<td>EE</td>
<td>Electronic Engineering</td>
<td>jarrows@wam</td>
</tr>
<tr>
<td>2</td>
<td>Peters</td>
<td>Kathy</td>
<td>HIST</td>
<td>History</td>
<td>kpeters2@wam</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Chris</td>
<td>HIST</td>
<td>History</td>
<td>smith2002@glue</td>
</tr>
<tr>
<td>4</td>
<td>Smith</td>
<td>John</td>
<td>CLIS</td>
<td>Information Studies</td>
<td>js03@wam</td>
</tr>
</tbody>
</table>
Problems with Join

• Data modeling for join is complex
  – Useful to start with E-R modeling

• Join are expensive to compute
  – Both in time and storage space

• But it is joins that make databases relational
  – Projection and restriction also used in flat files
Some Lingo

• “Primary Key” uniquely identifies a record
  – e.g. student ID in the student table

• “Compound” primary key
  – Synthesize a primary key with a combination of fields
  – e.g., Student ID + Course ID in the enrollment table

• “Foreign Key” is primary key in the other table
  – Note: it need not be unique in this table
## Project

### New Table

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Department ID</th>
<th>Department</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arrows</td>
<td>John</td>
<td>EE</td>
<td>Electronic Engineering</td>
<td>jarrows@wam</td>
</tr>
<tr>
<td>2</td>
<td>Peters</td>
<td>Kathy</td>
<td>HIST</td>
<td>History</td>
<td>kpeters2@wam</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Chris</td>
<td>HIST</td>
<td>History</td>
<td>smith2002@glue</td>
</tr>
<tr>
<td>4</td>
<td>Smith</td>
<td>John</td>
<td>CLIS</td>
<td>Information Stuides</td>
<td>js03@wam</td>
</tr>
</tbody>
</table>

### SQL Query

```sql
SELECT Student ID, Department
FROM
```
### New Table

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>Department ID</th>
<th>Department</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arrows</td>
<td>John</td>
<td>EE</td>
<td>Electronic Engineering</td>
<td>jarrows@wam</td>
</tr>
<tr>
<td>2</td>
<td>Peters</td>
<td>Kathy</td>
<td>HIST</td>
<td>History</td>
<td>kpeters2@wam</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Chris</td>
<td>HIST</td>
<td>History</td>
<td>smith2002@glue</td>
</tr>
<tr>
<td>4</td>
<td>Smith</td>
<td>John</td>
<td>CLIS</td>
<td>Information Studies</td>
<td>js03@wam</td>
</tr>
</tbody>
</table>

WHERE Department ID = “HIST”
Entity-Relationship Diagrams

- Graphical visualization of the data model
- Entities are captured in boxes
- Relationships are captured using arrows
Registrar ER Diagram

**Enrollment**
- Student
- Course
- Grade

**Student**
- Student ID
- First name
- Last name
- Department
- E-mail

**Course**
- Course ID
- Course Name

**Department**
- Department ID
- Department Name

has

associated with
Getting Started with E-R Modeling

• What **questions** must you answer?

• What **data** is needed to generate the answers?
  – Entities
    • Attributes of those entities
  – Relationships
    • Nature of those relationships

• How will the user interact with the system?
  – Relating the question to the available data
  – Expressing the answer in a useful form
Project Team E-R Example

- **student** is a **member-of** **team**.
- A **human** **needs** a **client**.
- The client **needs** a **project**.
- The project **creates** a **manage-role**, which is the role of **manage**, 1-to-1 relationship.
- The project also **creates** a **implement-role**, which is the role of **implement**, **M**-to-1 relationship.
- A **client** is related to a **team** through a **M**-to-1 relationship.
- A **team** can have **php-project** and **ajax-project**.
Components of E-R Diagrams

• Entities
  – Types
    • Subtypes (disjoint / overlapping)
  – Attributes
    • Mandatory / optional
  – Identifier

• Relationships
  – Cardinality
  – Existence
  – Degree
Types of Relationships

- Many-to-Many
- 1-to-Many
- 1-to-1
Making Tables from E-R Diagrams

• Pick a primary key for each entity
• Build the tables
  – One per entity
  – Plus one per M:M relationship
  – Choose terse but memorable table and field names
• Check for parsimonious representation
  – Relational “normalization”
  – Redundant storage of computable values
• Implement using a DBMS
• 1NF: Single-valued indivisible (atomic) attributes
  – Split “Doug Oard” to two attributes as (“Doug”, “Oard”)
  – Model M:M implement-role relationship with a table

• 2NF: Attributes depend on complete primary key
  – (id, impl-role, name)->(id, name)+(id, impl-role)

• 3NF: Attributes depend directly on primary key
  – (id, addr, city, state, zip)->(id, addr, zip)+(zip, city, state)

• 4NF: Divide independent M:M tables
  – (id, role, courses) -> (id, role) + (id, courses)

• 5NF: Don’t enumerate derivable combinations
Normalized Table Structure

- Persons: id, fname, lname, userid, password
- Contacts: id, ctype, cstring
- Ctlables: ctype, string
- Students: id, team, mrole
- Iroles: id, irole
- Rlabels: role, string
- Projects: team, client, pstring
A More Complex ER Diagram

cadastral: a public record, survey, or map of the value, extent, and ownership of land as a basis of taxation.

Source: US Dept. Interior Bureau of Land Management, Federal Geographic Data Committee Cadastral Subcommittee
http://www.fairview-industries.com/standardmodule/cad-erd.htm
Database Integrity

• Registrar database must be internally consistent
  – Enrolled students must have an entry in student table
  – Courses must have a name

• What happens:
  – When a student withdraws from the university?
  – When a course is taken off the books?
Integrity Constraints

• Conditions that must always be true
  – Specified when the database is designed
  – Checked when the database is modified

• RDBMS ensures integrity constraints are respected
  – So database contents remain faithful to real world
  – Helps avoid data entry errors
Referential Integrity

• Foreign key values must exist in other table
  – If not, those records cannot be joined

• Can be enforced when data is added
  – Associate a primary key with each foreign key

• Helps avoid erroneous data
  – Only need to ensure data quality for primary keys
Database “Programming”

• Natural language
  – Goal is ease of use
    • e.g., Show me the last names of students in CLIS
  – Ambiguity sometimes results in errors

• Structured Query Language (SQL)
  – Consistent, unambiguous interface to any DBMS
  – Simple command structure:
    • e.g., SELECT Last name FROM Students WHERE Dept=CLIS
  – Useful standard for inter-process communications

• Visual programming (e.g., Microsoft Access)
  – Unambiguous, and easier to learn than SQL
The SELECT Command

• Project chooses columns
  – Based on their label

• Restrict chooses rows
  – Based on their contents
    • e.g. department ID = “HIST”

• These can be specified together
  – SELECT Student ID, Dept WHERE Dept = “History”
Restrict Operators

• Each SELECT contains a single WHERE

• Numeric comparison
  \(<, >, =, <>\), ...
  • e.g., grade<80

• Boolean operations
  – e.g., Name = “John” AND Dept <> “HIST”
Databases in the Real World

• Some typical database applications:
  – Banking (e.g., saving/checking accounts)
  – Trading (e.g., stocks)
  – Airline reservations

• Characteristics:
  – Lots of data
  – Lots of concurrent access
  – Must have fast access
  – “Mission critical”
Concurrency

• Thought experiment: You and your project partner are editing the same file...
  – Scenario 1: you both save it at the same time
  – Scenario 2: you save first, but before it’s done saving, your partner saves

Whose changes survive?
A) Yours  B) Partner’s  C) neither  D) both  E) ???
Concurrent Example

• Possible actions on a checking account
  – Deposit check (read balance, write new balance)
  – Cash check (read balance, write new balance)

• Scenario:
  – Current balance: $500
  – You try to deposit a $50 check and someone tries to cash a $100 check at the same time
  – Possible sequences: (what happens in each case?)

- Deposit: read balance
  Deposit: write balance
  Cash: read balance
  Cash: write balance

- Deposit: read balance
  Deposit: write balance
  Cash: read balance
  Deposit: write balance

- Deposit: read balance
  Deposit: write balance
  Cash: read balance
  Cash: write balance

- Deposit: read balance
  Cash: read balance
  Deposit: write balance
  Cash: write balance
Database Transactions

• Transaction: sequence of grouped database actions
  – e.g., transfer $500 from checking to savings

• “ACID” properties
  – Atomicity
    • All-or-nothing
  – Consistency
    • Each transaction must take the DB between consistent states.
  – Isolation:
    • Concurrent transactions must appear to run in isolation
  – Durability
    • Results of transactions must survive even if systems crash
Making Transactions

• Idea: keep a log (history) of all actions carried out while executing transactions
  – Before a change is made to the database, the corresponding log entry is forced to a safe location

• Recovering from a crash:
  – Effects of partially executed transactions are undone
  – Effects of committed transactions are redone
Key Ideas

• Databases are a good choice when you have
  – Lots of data
  – A problem that contains inherent relationships

• Design before you implement
  – This is just another type of programming
  – The mythical person-month applies!

• Join is the most important concept
  – Project and restrict just remove undesired stuff
RideFinder Exercise

• Design a database to match passengers with available rides for Spring Break
  – Drivers phone in available seats
    • They want to know about interested passengers
  – Passengers call up looking for rides
    • They want to know about available rides
    • No “ride wanted” ads
  – These things happen in no particular order
Exercise Goals

- Identify the tables you will need
  - First decide what data you will need
    - What questions will be asked?
  - Then design normalized tables
    - Start with binary relations if that helps

- Design the queries
  - Using join, project and restrict
  - What happens when a passenger calls?
  - What happens when a driver calls?
Reminder: Starting E-R Modeling

• What **questions** must you answer?

• What **data** is needed to generate the answers?
  – Entities
    • Attributes of those entities
  – Relationships
    • Nature of those relationships

• How will the user interact with the system?
  – Relating the question to the available data
  – Expressing the answer in a useful form
Exercise Logistics

• Work in groups of 3 or 4
• Brainstorm data requirements for 5 minutes
  – Do passengers care about the price?
  – Do drivers care how much luggage there is?
• Develop tables and queries for 15 minutes
  – Don’t get hung up on one thing too long
• Compare you answers with another group
  – Should take about 5 minutes each
Making Tables from E-R Diagrams

- Pick a primary key for each entity
- Build the tables
  - One per entity
  - Plus one per M:M relationship
  - Choose terse but memorable table and field names
- Check for parsimonious representation
  - Relational “normalization”
  - Redundant storage of computable values
- Implement using a DBMS
• 1NF: Single-valued indivisible (atomic) attributes
  – Split “Doug Oard” to two attributes as (“Doug”, “Oard”)
  – Model M:M implement-role relationship with a table

• 2NF: Attributes depend on complete primary key
  – (id, impl-role, name) -> (id, name) + (id, impl-role)

• 3NF: Attributes depend directly on primary key
  – (id, addr, city, state, zip) -> (id, addr, zip) + (zip, city, state)

• 4NF: Divide independent M:M tables
  – (id, role, courses) -> (id, role) + (id, courses)

• 5NF: Don’t enumerate derivable combinations
Using Microsoft Access

• Create a database called M:\rides.mdb
  – File->New->Blank Database

• Specify the fields (columns)
  – “Create a Table in Design View”

• Fill in the records (rows)
  – Double-click on the icon for the table
Creating Fields

• Enter field name
  – Must be unique, but only within the same table

• Select field type from a menu
  – Use date/time for times
  – Use text for phone numbers

• Designate primary key (right mouse button)

• Save the table
  – That’s when you get to assign a table name
Entering Data

• Open the table
  – Double-click on the icon

• Enter new data in the bottom row
  – A new (blank) bottom row will appear

• Close the table
  – No need to “save” – data is stored automatically
Building Queries

• Copy ride.mdb to your M:\ drive

• “Create Query in Design View”
  – In “Queries”

• Choose two tables, Flight and Company

• Pick each field you need using the menus
  – Unclick “show” to not project
  – Enter a criterion to “restrict”

• Save, exit, and reselect to run the query
Some Details About Access

• Joins are automatic if field names are same
  – Otherwise, drag a line between the fields

• Sort order is easy to specify
  – Use the menu

• Queries form the basis for reports
  – Reports give good control over layout
  – Use the report wizard - the formats are complex

• Forms manage input better than raw tables
  – Invalid data can be identified when input
  – Graphics can be incorporated
Databases in the Real World

• Some typical database applications:
  – Banking (e.g., saving/checking accounts)
  – Trading (e.g., stocks)
  – Airline reservations

• Characteristics:
  – Lots of data
  – Lots of concurrent access
  – Must have fast access
  – “Mission critical”
**Caching servers:** 15 million requests per second, 95% handled by memcache (15 TB of RAM)

**Database layer:** 800 eight-core Linux servers running MySQL (40 TB user data)
Database Integrity

• Registrar database must be internally consistent
  – Enrolled students must have an entry in student table
  – Courses must have a name

• What happens:
  – When a student withdraws from the university?
  – When a course is taken off the books?
Integrity Constraints

- Conditions that must always be true
  - Specified when the database is designed
  - Checked when the database is modified

- RDBMS ensures integrity constraints are respected
  - So database contents remain faithful to real world
  - Helps avoid data entry errors
Referential Integrity

• Foreign key values must exist in other table
  – If not, those records cannot be joined

• Can be enforced when data is added
  – Associate a primary key with each foreign key

• Helps avoid erroneous data
  – Only need to ensure data quality for primary keys
Concurrenty

- Thought experiment: You and your project partner are editing the same file...
  - Scenario 1: you both save it at the same time
  - Scenario 2: you save first, but before it’s done saving, your partner saves

  Whose changes survive?
  A) Yours  B) Partner’s  C) neither  D) both  E) ???
Concurrent Example

• Possible actions on a checking account
  – Deposit check (read balance, write new balance)
  – Cash check (read balance, write new balance)

• Scenario:
  – Current balance: $500
  – You try to deposit a $50 check and someone tries to cash a $100 check at the same time
  – Possible sequences: (what happens in each case?)

  Deposit: read balance
  Deposit: write balance
  Cash: read balance
  Cash: write balance

  Deposit: read balance
  Cash: read balance
  Deposit: write balance

  Deposit: write balance
  Cash: read balance
  Cash: write balance
Database Transactions

• Transaction: sequence of grouped database actions
  – e.g., transfer $500 from checking to savings

• “ACID” properties
  – Atomicity
    • All-or-nothing
  – Consistency
    • Each transaction must take the DB between consistent states.
  – Isolation:
    • Concurrent transactions must appear to run in isolation
  – Durability
    • Results of transactions must survive even if systems crash
Making Transactions

• Idea: keep a log (history) of all actions carried out while executing transactions
  – Before a change is made to the database, the corresponding log entry is forced to a safe location

• Recovering from a crash:
  – Effects of partially executed transactions are undone
  – Effects of committed transactions are redone
Databases in Web Applications
Why Database-Generated Pages?

• Remote access to a database
  – Client does not need the database software

• Serve rapidly changing information
  – e.g., Airline reservation systems

• Provide multiple “access points”
  – By subject, by date, by author, …

• Record user responses in the database
Issues to Consider

• Benefits
  – Multiple views
  – Data reuse
  – Scalable
  – Access control

• Costs
  – Formal modeling
  – Complex (learn, design, implement, debug)
  – Brittle (relies on multiple communicating servers)
  – Not crawlable
Putting the Pieces Together
Structured Query Language

DESCRIBE Flight;
Structured Query Language

SELECT * FROM Flight;
Structured Query Language

FROM Flight, Company
WHERE Flight.CompanyName=Company.CompanyName
AND Flight.AvailableSeats>3;
select address from employee
where employee.surname='Smith' and employee.forenames='Robert';

how you want to restrict the rows
```
select dname
from employee, department
where employee.depno=department.depno
and surname='Smith' and forenames='Robert';
```
Create a MySQL Database

• “root” user creates database + grants permissions
  – Using the WAMP console (or mysql –u root –p)
    • root has no initial password; just hit <enter> when asked
  – By the system administrator on OTAL (otal.umd.edu)
    CREATE DATABASE project;
    GRANT SELECT, INSERT, UPDATE, DELETE, INDEX, ALTER, CREATE, DROP ON project.* TO 'foo'@'localhost' IDENTIFIED BY 'bar';
    FLUSH PRIVILEGES;

• Start mysql
  – MySQL console for WAMP, ssh for OTAL
    mysql –u foo –p bar

• Connect to your database
  USE project;
Creating Tables

```
CREATE TABLE contacts (  
  ckey    MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT,  
  id      MEDIUMINT UNSIGNED NOT NULL,  
  ctype   SMALLINT UNSIGNED NOT NULL,  
  cstring VARCHAR(40) NOT NULL,  
  FOREIGN KEY (id) REFERENCES persons(id) ON DELETE CASCADE,  
  FOREIGN KEY (ctype) REFERENCES ctlabels(ctype) ON DELETE RESTRICT,  
  PRIMARY KEY (ckey)  
) ENGINE=INNODB;
```

➢ To delete: DROP TABLE contacts;
Populating Tables

INSERT INTO ctlabels
    (string) VALUES
    ('primary email'),
    ('alternate email'),
    ('home phone'),
    ('cell phone'),
    ('work phone'),
    ('AOL IM'),
    ('Yahoo Chat'),
    ('MSN Messenger'),
    ('other');

➢ To empty a table: DELETE FROM ctlabels;
“Looking Around” in MySQL

- SHOW DATABASES;
- SHOW TABLES;
- DESCRIBE tablename;
- SELECT * FROM tablename;
SQL Utility Service Desk Exercise

• Design a database to keep track of service calls for a utility company:
  – Customers call to report problems
  – Call center manages “tickets” to assign workers to jobs
    • Must match skills and service location
    • Must balance number of assignments
  – Workers call in to ask where their next jobs are

• In SQL, you can do the following operations:
  – Count the number of rows in a result set
  – Sort the result set according to a field
  – Find the maximum and minimum value of a field
Before You Go

On a sheet of paper, answer the following (ungraded) question (no names, please):

What was the muddiest point in today’s class?