College of Information Studies

University of Maryland Hornbake Library Building College Park, MD 20742-4345

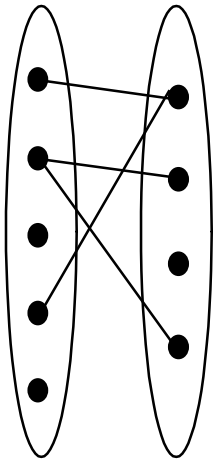
Relational Databases

Week 11

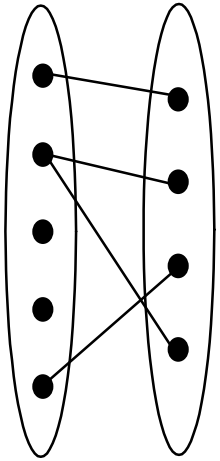
LBSC 671

Creating Information Infrastructures

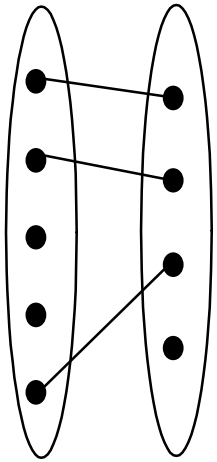
Types of Relationships



Many-to-Many

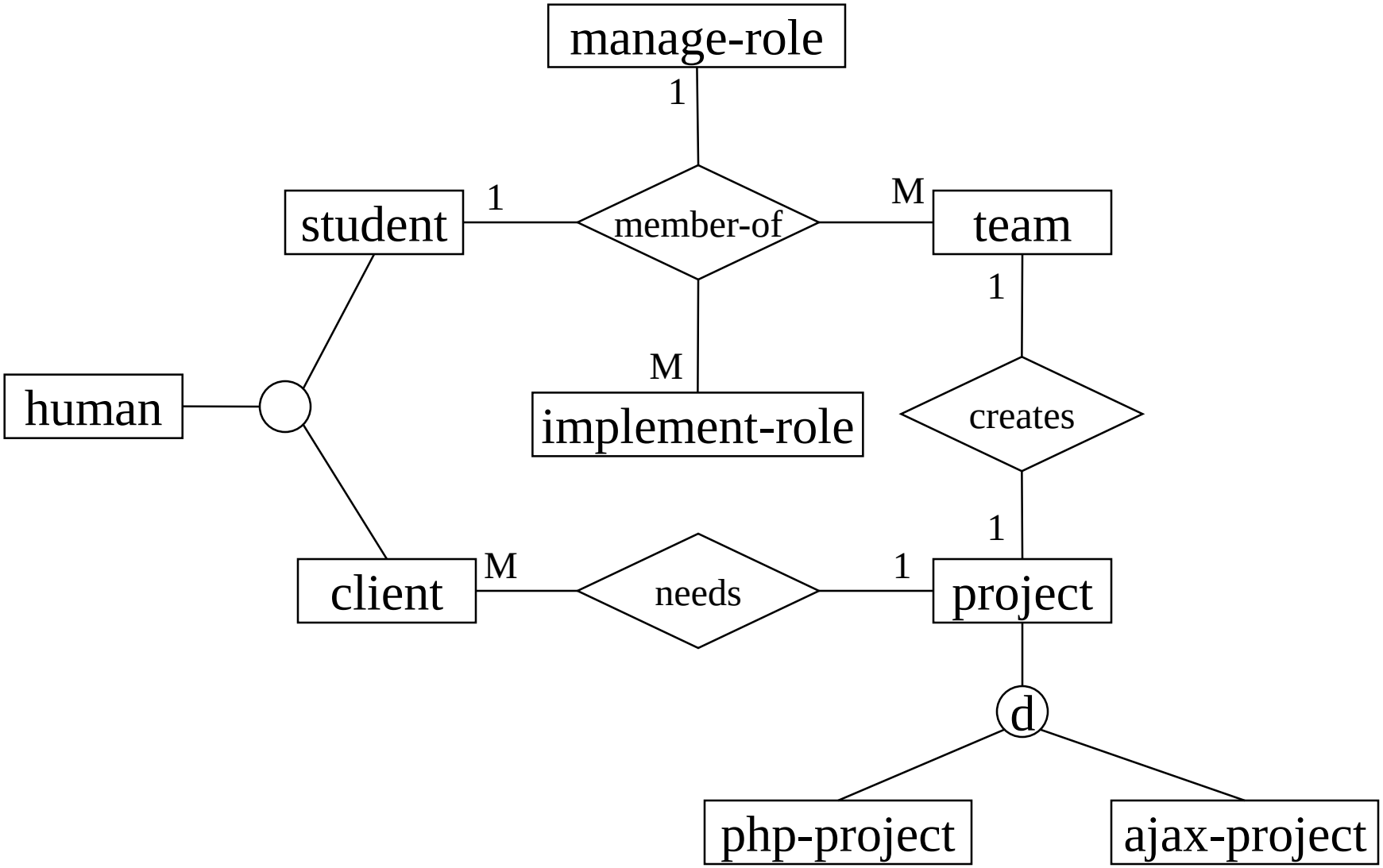


1-to-Many



1-to-1

Project Team E-R Example



Normalized Table Structure

- Persons: id, fname, lname, userid, password
- Contacts: id, ctype, cstring
- Ctlabels: ctype, string
- Students: id, team, mrole
- Iroles: id, irole
- Rlabels: role, string
- Projects: team, client, pstring

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”

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

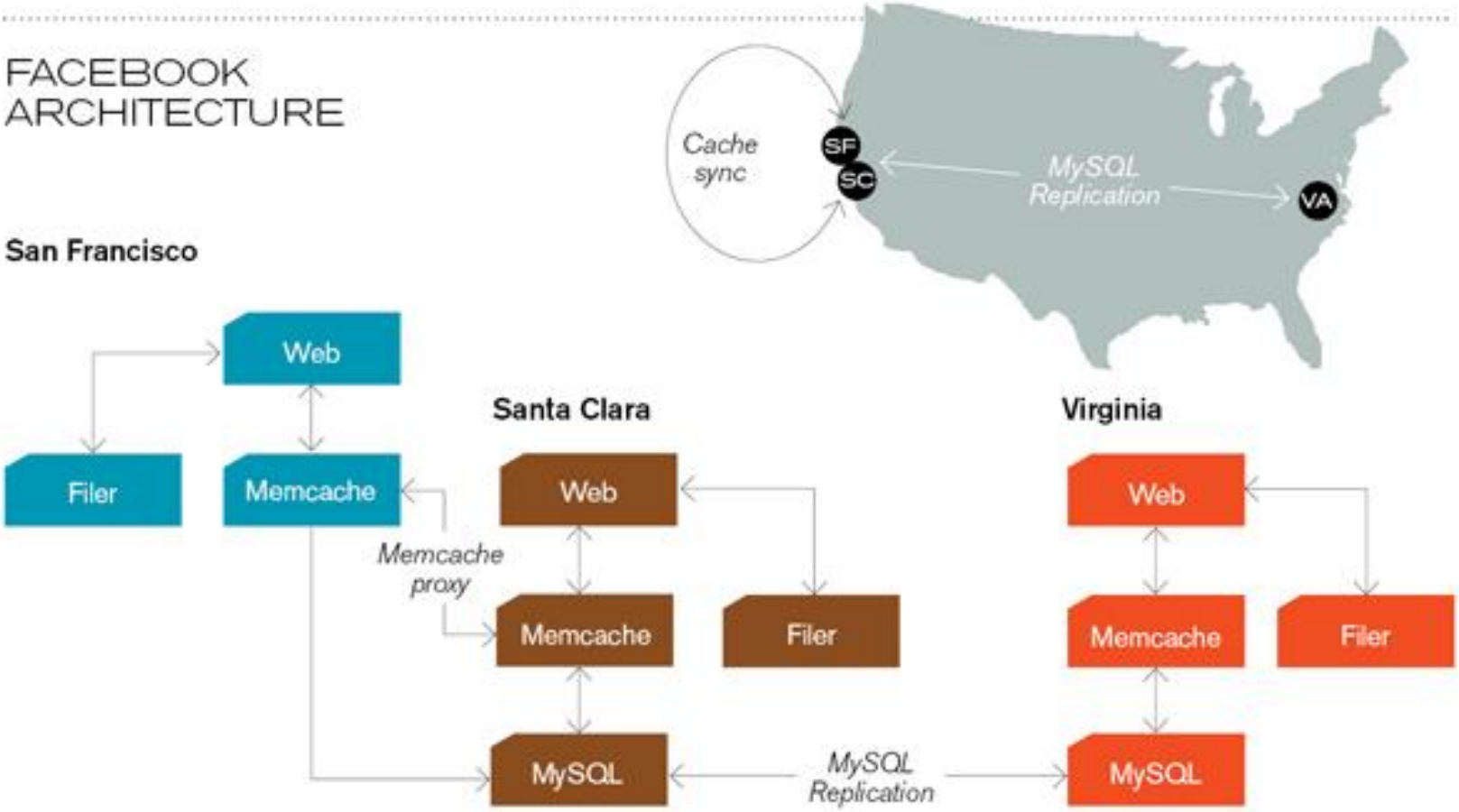
Some Details About Access

- 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)

Source: Technology Review (July/August, 2008)

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

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) ???

Concurrency 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
Cash: write balance
Deposit: write balance

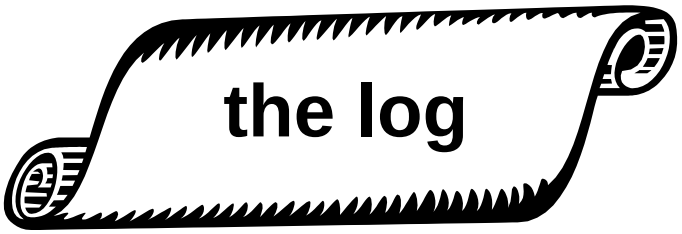
Deposit: read balance
Cash: read balance
Deposit: write balance
Cash: write balance

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

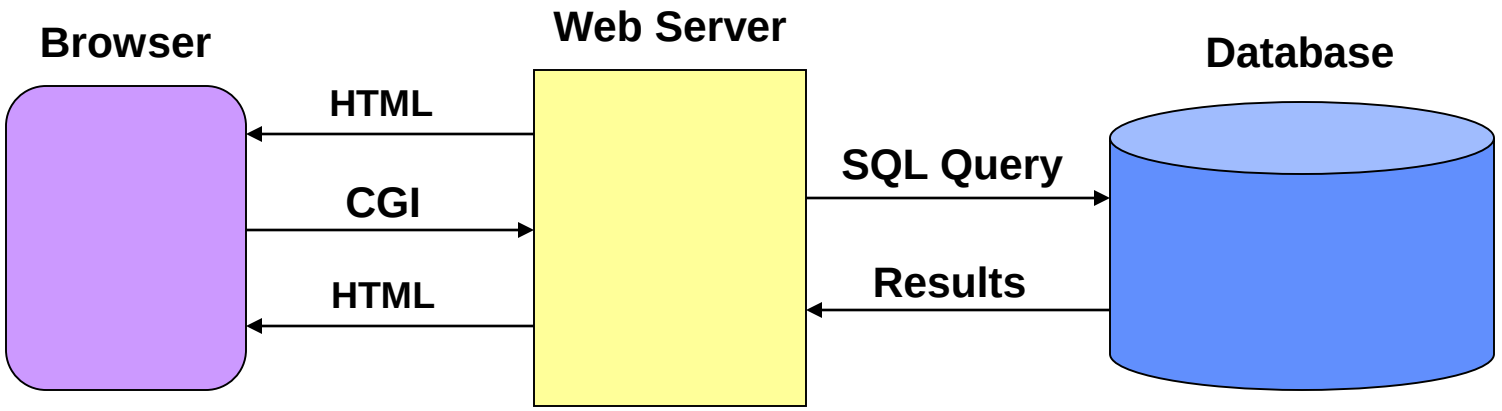
Making Transactions

- 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

Putting the Pieces Together

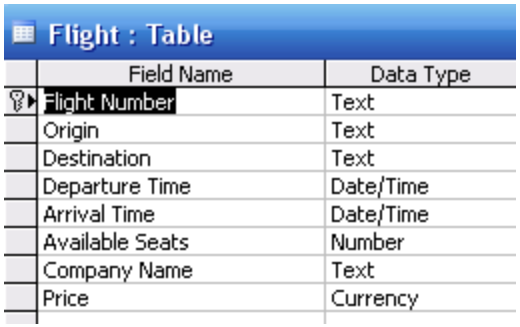


Why Database-Generated Pages?

- 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

Structured Query Language

DESCRIBE Flight;



Flight : Table

Field Name	Data Type
Flight Number	Text
Origin	Text
Destination	Text
Departure Time	Date/Time
Arrival Time	Date/Time
Available Seats	Number
Company Name	Text
Price	Currency

Structured Query Language

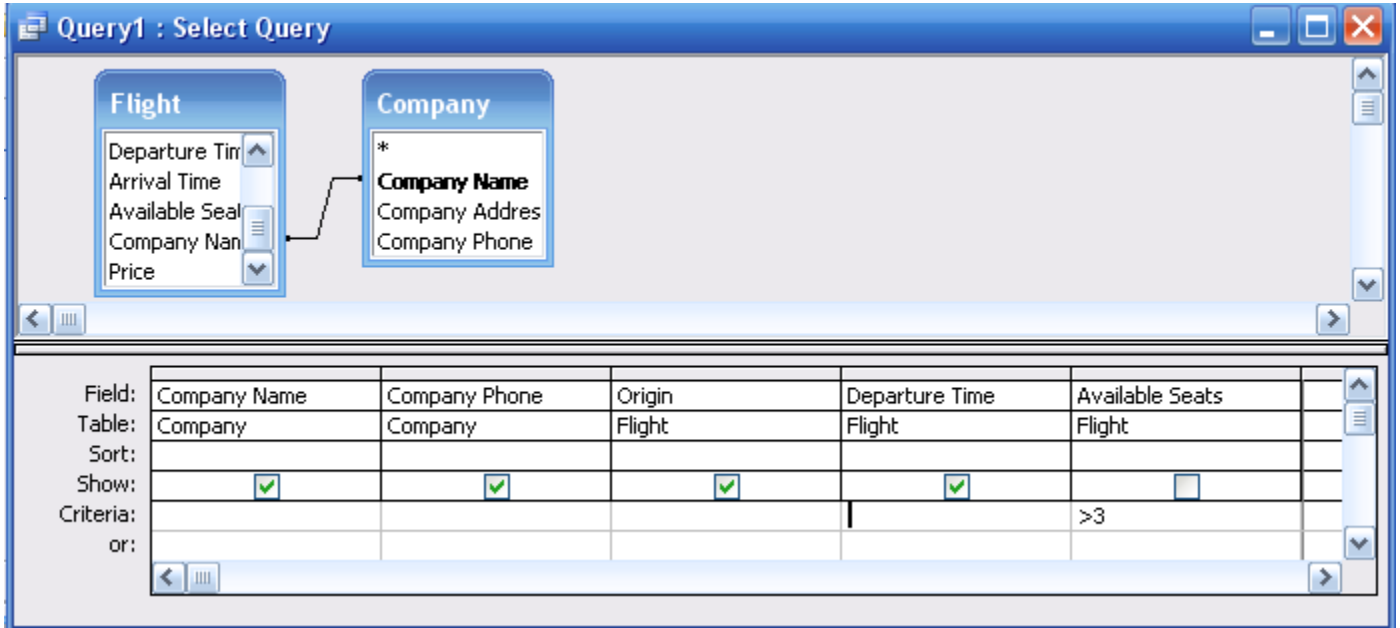
SELECT * FROM Flight;

Flight : Table

Flight Number	Origin	Destination	Departure Time	Arrival Time	Available Seats	Company Name	Price
CA210	DC	Austin	6:00:00 AM	11:00:00 AM	0	Cal Air	$200.00
CA345	San Jose	San Diego	9:00:00 AM	10:30:00 AM	20	Cal Air	$100.00
FT900	Chicago	New York	2:00:00 PM	5:00:00 PM	1	Fancy Trans	$200.00
GJ405	DC	San Jose	12:30:00 PM	8:45:00 PM	10	Green Jet	$340.00
GJ908	New York	Austin	8:00:00 AM	12:00:00 PM	2	Green Jet	$250.00
TP123	New York	San Jose	7:00:00 AM	11:00:00 AM	2	Trans Planet	$400.00
					0		$0.00

Structured Query Language

```
SELECT Company.CompanyName, Company.CompanyPhone,
    Flight.Origin, Flight.DepartureTime
FROM Flight,Company
WHERE Flight.CompanyName=Company.CompanyName
  AND Flight.AvailableSeats>3;
```



Issues to Consider

- Benefits of Databases
  - Multiple views
  - Data reuse
  - Scalable
  - Access control
- Costs of Databases
  - Formal modeling
  - Complex (learn, design, implement, debug)
  - Brittle (relies on multiple communicating servers)
  - Not crawlable

Key Ideas

- Databases are a good choice when you have
  - Lots of data
  - A problem that contains inherent relationships
- Design before you implement
- 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

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?