College of Information Studies

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

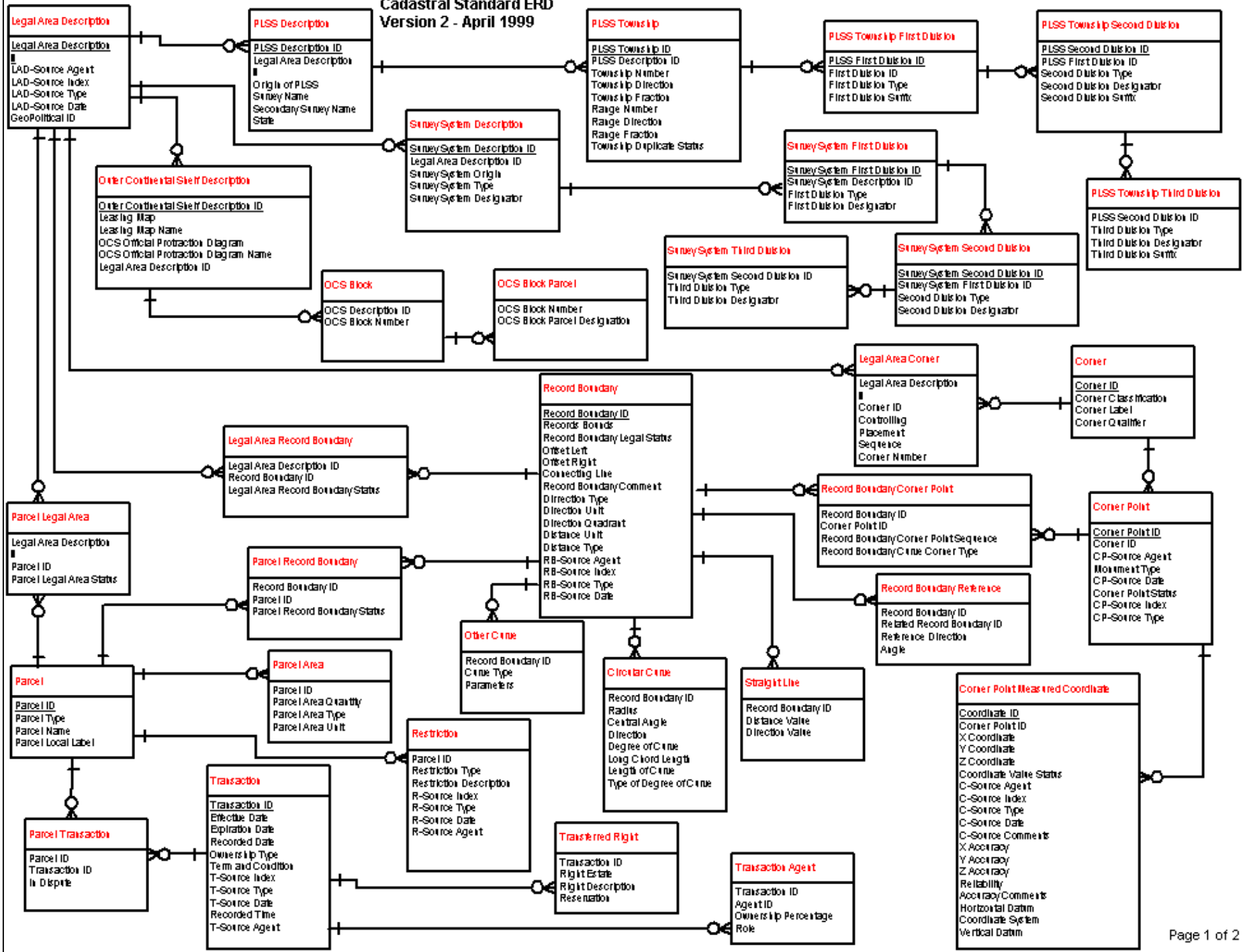
Relational Databases (Part 2)

Session 14

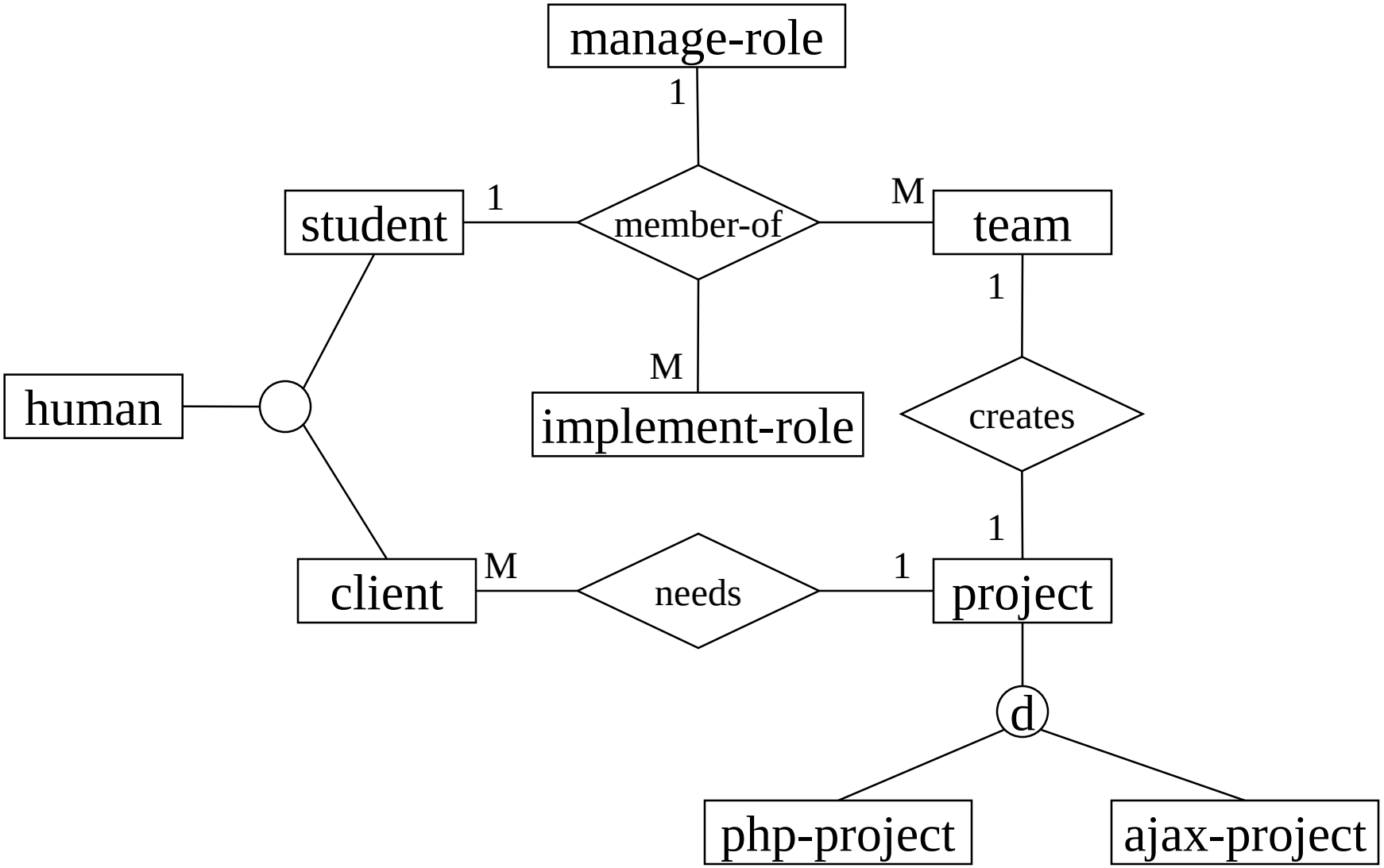
INST 301

Introduction to Information Science

Cadastral Standard ERD
Version 2 - April 1999



“Project Team” E-R Example



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

One Possible 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

```
CREATE TABLE persons (
  id      MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT,
  fname   VARCHAR(15) NOT NULL,
  lname   VARCHAR(30) NOT NULL,
  userid  VARCHAR(40) NOT NULL,
  password VARCHAR(40) NOT NULL,
  PRIMARY KEY (id)
) ENGINE=INNODB;

CREATE TABLE ctlabels (
  ctype   SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  string  VARCHAR(40) NOT NULL,
  PRIMARY KEY (ctype)
) ENGINE=INNODB;

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;

CREATE TABLE rlabels (
  role    SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  string  VARCHAR(40) NOT NULL,
  PRIMARY KEY (role)
) ENGINE=INNODB;

CREATE TABLE projects (
  team    SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  client  MEDIUMINT UNSIGNED NOT NULL,
  string  VARCHAR(40) NOT NULL,
  FOREIGN KEY (client) REFERENCES persons(id) ON DELETE RESTRICT,
  PRIMARY KEY (team)
) ENGINE=INNODB;

CREATE TABLE students (
  id      MEDIUMINT UNSIGNED NOT NULL,
  team    SMALLINT UNSIGNED,
  mrole   SMALLINT UNSIGNED,
  FOREIGN KEY (id) REFERENCES persons(id) ON DELETE CASCADE,
  FOREIGN KEY (team) REFERENCES projects(team) ON DELETE SET NULL,
  FOREIGN KEY (mrole) REFERENCES rlabels(role) ON DELETE SET NULL,
  PRIMARY KEY (id)
) ENGINE=INNODB;
```

```
CREATE TABLE iroles (
  ikey    MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT,
  id      MEDIUMINT UNSIGNED NOT NULL,
  irole   SMALLINT UNSIGNED NOT NULL,
  FOREIGN KEY (id) REFERENCES persons(id) ON DELETE CASCADE,
  FOREIGN KEY (irole) REFERENCES rlabels(role) ON DELETE CASCADE,
  PRIMARY KEY (ikey)
) ENGINE=INNODB;

INSERT INTO rlabels
  (string) VALUES
  ('Project Manager'),
  ('System Architect'),
  ('Data Architect'),
  ('Test Designer'),
  ('PHP Programmer'),
  ('JavaScript Programmer'),
  ('Database Administrator'),
  ('XML Designer');

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

INSERT INTO persons
  (fname, lname, userid, password) VALUES
  ('Sam',      'Cooper',       'coopers',   SHA('pass')),
  ('Lindsey',  'Goshen',       'goshenl',   SHA('pass')),
  ('Tommy',    'Teller',       'tellert',   SHA('pass')),
  ('Nancy',    'Lange',        'langen',    SHA('pass'));
```

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

Exercise Steps

- Create an E-R model
  - What entities?
  - What attributes?
  - What relations?
- Build the Tables
  - One per entity
  - Plus one per many-to-many relation
- Build the Queries
  - What happens when a passenger calls?
  - What happens when a driver calls?

Exercise Logistics

- Work in dissimilar teams of 2
- Build an E-R model (10 minutes)
  - One team will present theirs to the class (5 min)
- Create the database (25 minutes)
  - Create tables
  - Put data in the tables
  - Create the queries
- One team will demo theirs to the class (5 min)

- 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

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?

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

- 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

Key Ideas

- Databases are a good choice when you have
  - Lots of data
  - A problem that contains inherent relationships
- Join is the most important concept
  - Project and restrict just remove undesired stuff
- Design before you implement
  - Managing complexity is important