```
-- CSE 344 Lecture 02 -- Relational Data Model
-- Reading: 2.1, 2.2, 2.3
--
-------------------------------------------------------------------
----------
-- 1. Review: Fundamental Concepts
--
-- We want to store data in a database to get all the great features
we
-- discussed last lecture (an efficient way to ask all sorts of
questions
-- on the data, an easy way to update the data, recovery from crashes,
-- etc.)

-- We need to somehow tell the database management system (DBMS) about
-- our data. For example, we may want to create a database of movies.
-- We have information about movies, actors, and casts. We need to
-- somehow tell the DBMS about this data.

-- A DBMS allows a user to define the data stored in terms of a data
model.

-- A data model is a general, conceptual way of structuring data

-- 1.1 Data model:  general, conceptual way of structuring data
--
-- Data models studied in this course:
--      relational data model  -- data = relations
--      semistructured data model (XML)  -- data = a tree
--
-- Other data models:
--      key-value pairs -- used by the NoSQL systems
--      graph data model -- used by RDF
--      object-oriented -- often used as a layer over relational model
--
--
-- Note: a data model describes both the data AND a query language
--
-- 1.2 Schema --  the structure of a particular database under a
certain data model
--
-- 1.3 Instance -- the actual data
--
-------------------------------------------------------------------
-- 2. The relational Data Model
--   Database instance:
--   -- "table" or "relation"
--   -- "column" or "attribute" or "field"
--   -- "row" or "tuple" or "record"
--
```

```
--   Cardinality of relation instance: nb tuples

--   Database schema:
--   -- "table name" or "relation name"
--   -- "column name" or "attribute name"
--   -- each attribute has a "type" or "domain"
--
--   Degree (or arity) of relation: nb attributes
--
--   Types or datatypes
--
--   -- Character strings: CHAR(20), VARCHAR(50), TEXT
--   -- Numbers: INT, BIGINT, SMALLINT, FLOAT
--   -- Others: MONEY, DATETIME,
--
--   -- Types are vendor specific
--   -- Types are static and strictly enforced;
--         exception: sqlite has dynamic types
--         http://www.sqlite.org/datatype3.html
--
--   Keys:
--   an attribute is called a "key" if it uniquely identifies a record
--
--   We can have a key with multiple attributes: what does this mean ?
--   It means that each unique combination of values for those
attributes
--   uniquely determines the record.

--   Candidate key: Minimal set of fields that uniquely identify
--   each tuple in a relation (candidate key = key)
--   Primary key: One candidate key can be selected as primary key
--
--   Foreign keys:
--   other tuples may use key values as "logical pointers"
--
--
--
-- Example on Whiteboard:
--
--   w/o types: Company(cname, country, no_employees, for_profit)
--   w/ types:  Company(cname: varchar(30), country: char(20),
no_employees:int, for_profit:char(1))
--   specify keys by underlining
--
-- Note:  if we had a semistructured data model, we would create
-- a tree. We would say: At the root, there is a company. Below it,
there are
-- departments. Each department has employees as its children, etc.
--
-- A database instance is the actual content of the tables in the
```

```
database.
--
-------------------------------------------------------------------------
----------
-- 3. The Relational Data Model in SQL
--
-- We will use SQLite in class
--
--
-- 3.1 Creating tables
--
create table Company
  (cname varchar(20) primary key,
   country varchar(20),
   no_employees int,
   for_profit char(1));

insert into Company  values ('GizmoWorks', 'USA',  20000,'y');
insert into Company  values ('Canon',     'Japan', 50000,'y');
insert into Company  values ('Hitachi',   'Japan', 30000,'y');
insert into Company  values('Charity',    'Canada',  500,'n');

select * from Company;

-- Making sure SQL Lite shows us the data in a nicer format
-- These commands are specific to SQLite!
.header on
.mode column
.nullvalue NULL

--
--
-- Comment: upper/lower case; name conflicts
--    -- Company, company, COMPANY  = all the same
--    -- Company(cname, country), Person(pname, country) = repeated
'country' OK
--    -- Company(cname, country), Person(pname, company) = the
attribute 'company' not ok

-- Null values: whenever we don't know the value, we can set it to
NULL

insert into Company values('MobileWorks', 'China', null, null);
select * from Company;

-- Deleting tuples from the database:

delete from Company where cname = 'Hitachi';
select * from Company;
```

```
delete from Company where for_profit = 'n';
-- what happens here??

select * from Company;

-- note: sql lite is REALLY light: it accepts many erroneous commands,
-- which other RDBMS would not accept.  We will flag these as alerts.

-- Alert 1: sqlite allows a key to be null

insert into Company values(NULL, 'Somewhere', 0, 'n');
select * from Company;

-- this is dangerous, since we cannot uniquely identify the tuple
-- better delete it before we get into trouble

delete from Company where country = 'Somewhere';
select * from Company;

-- Discussion in class:
--   tables are NOT ordered. They represent sets or bags.
--   tables do NOT prescribe how they should be implemented: PHYSICAL
DATA INDEPENDENCE!
--   tables are FLAT (all attributes are base types)

-- Discussion: how would you implement a table?
--   row oriented
--   column oriented
--   vertically partitioned
--   horizontally partitioned
-- What are the pros/cons of the different physical implementations ?

-- Why is physical data independence important?
-- It is important so that we can optimize the data layout on disk for
-- performance without breaking the applications written on top of the
database!


-- 3.2  Altering a table  in SQL

-- Add/Drop attribute(s)
-- let's drop the for_profit attribute:

-- Note: SQL Lite does not support dropping an attribute:
-- ALTER TABLE Company DROP for_profit;  -- doesn't work

ALTER TABLE Company ADD ceo varchar(20);
select * from Company;

UPDATE Company SET ceo='Brown' WHERE cname = 'Canon';
```

```
SELECT * FROM Company;

-- A peek at the physical implementation:

-- What happens when you alter a table ?  Consider row-wise and
column-wise.

-- 3.3 Multiple Tables, and Keys - Foreign Keys
-- Now alter Company to add the products that they manufacture.
-- Problem: can't add an attribute that is a LIST OF PRODUCTS. What
should we do??
--
--

-- Create a separate table Product, with a foreign key to the company:
create table Product
  (pname varchar(20) primary key,
   price float,
   category varchar(20),
   manufacturer varchar(20) references Company);

-- Alert 2: sqlite does NOT enforce foreign keys by default. To enable
-- foreign keys use the following command. The command will have no
-- effect if your version of SQLite was not compiled with foreign keys
-- enabled. Do not worry about it.

PRAGMA foreign_keys=ON;

insert into Product values('Gizmo',      19.99, 'gadget',
'GizmoWorks');
insert into Product values('PowerGizmo', 29.99, 'gadget',
'GizmoWorks');
insert into Product values('SingleTouch', 149.99, 'photography',
'Canon');
insert into Product values('MultiTouch', 199.99, 'photography',
'MobileWorks');
insert into Product values('SuperGizmo', 49.99, 'gadget',
'MobileWorks');

select * from Product;

-- If we try:
insert into Product values('MultiTouch2', 199.99, 'photography',
'H2');
-- We should get an error if foreign keys got enforced
-- Error: foreign key constraint failed
```