```
-- CSE 344 Lecture 03 -- Basic SQL
-- Readings: 6.1, 6.2
-------------------------------------------------------------------------------
-- we will use the following schema in this lecture:
-- Product(pname, price, category, manufacturer)
-- Company(cname, country)

-------------------------------------------------------------------------------
create table Company
  (cname varchar(20) primary key,
   country varchar(20));

insert into Company  values ('GizmoWorks', 'USA');
insert into Company  values ('Canon',    'Japan');
insert into Company  values ('Hitachi',  'Japan');

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

-- Alert 1: As mentioned in the lecture 2 notes, sqlite allows a key to be null:

insert into Company values(NULL, 'Somewhere');

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

delete from Company where country = 'Somewhere';

-------------------------------------------------------------------------------
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', 'Hitachi');
insert into Product values('SuperGizmo', 49.99, 'gadget', 'Hitachi');

-- 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

-------------------------------------------------------------------------------
-- Notice that the data we created is stored on disk.
-- Quite sqlite3
-- See that file "lecture3" on disk has now a non-zero size.
-- It's a binary file. It contains the data for all our relations in one file.
-- When we come back to sqlite3, all our data is there.

-------------------------------------------------------------------------------
-- 1. SELECTION queries select a subset of the table:

-- Before we start, let's switch to a better query output format
.mode column
.header ON
```

```
-- What do you think the following queries return?

select *
from Product
where price > 100.0;

select *
from Product
where pname like '%e%';


-------------------------------------------------------------------------------
-- 2. PROJECTION queries keep a subset of the attributes

select price, category
from Product;


-------------------------------------------------------------------------------
-- some minor variations: DISTINCT and ORDER BY

-- This query returns duplicates:
select category
from Product;

-- Wait a minute... didn't we say that relations were sets? Why
-- do we suddently see bags? Why isn't the DBMS eliminating duplicates?
-- Key reason is performance: eliminating duplicates is an expensive operations.
-- So the DBMS will leave them if the user/application can tolerate them.

-- To eliminate duplicates, use DISTINCT:

select distinct category
from Product;

-- We can also order the outputs using ORDER BY

-- order alphabetically by name:
select *
from product
order by pname;

-- order by price descending
select *
from product
order by price desc;

-- order by manufacturer, then price descenting
select *
from product
order by manufacturer, price desc;

-- What happens if we order on an attribute that we do NOT return ?

-- First, let's try:
select *
from Product
order by manufacturer;

-- Now, let's try:
select category
from Product
order by manufacturer;

-- What happens if we also do DISTINCT ?
-- The query should fail but...
-- Alert 3: sqlite does the wrong thing here, again:
```

```
select distinct category
from Product
order by manufacturer;


-------------------------------------------------------------------------------
-- 3. JOINS

-- What should the following query return?

select pname, price
from   Product, Company
where  manufacturer=cname and country='Japan' and price < 150;

-- Let's analyze it together on the white board.
-- Note that manufacturer=cname is called the "join predicate"

-------------------------------------------------------------------------------
-- ***** In class:
-- ***** Retreive all American companies that manufacture products in the 'gadget' category
--
--
--
--
--
--
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget'
AND manufacturer = cname;

-- **** Retreive all Japanese companies that manufacture products in
--      both the 'gadget' and the 'photography' categories
--
--
--
--
--
--
SELECT DISTINCT cname
FROM Product P1, Product P2, Company
WHERE country = 'Japan'
AND P1.category = 'gadget'
AND P2.category = 'photography'
AND P1.manufacturer = cname AND P2.manufacturer = cname;

-------------------------------------------------------------------------------
-- Joins may introduce duplicates:

select  country
from    Product, Company
where   manufacturer=cname and category='gadget';

-- easy fix:

select  distinct country
from    Product, Company
where   manufacturer=cname and category='gadget';

-------------------------------------------------------------------------------
-- Aliases = are tuple variables that allow us to disambiguate attribute names

-- find all countries that manufacture both a product under $25 and a product over $25

select distinct x.country
from Company x, Product y, Product z
where x.cname = y.manufacturer and y.price < 25
  and x.cname = z.manufacturer and z.price > 25;
```

```
-- when no aliases are given, then the table name is used as an alias


--------------------------------------------------------------------------------
-- The "nested loop" semantics of SQL queries
--
-- Query:
--    select a1, a2, ..., ak
--    from R1 as x1, R2 as x2, ...., Rm as xm
--    where Cond
--
-- Semantics:
--    for x1 in R1 do
--       for x2 in R2 do
--          for x3 in R3 do
--              ...
--                 for xn in Rm do
--                      if Cond then output(a1,...,ak)
--
--
--
-- However, the query processor will ALMOST NEVER evaluate the query this way !
--
-- ****** Important Concept: SQL IS A DELCARATIVE LANGUAGE
-- ****** What it means: In SQL we say WHAT we want
-- ******                the system figures out HOW to compute it
--------------------------------------------------------------------------------
-- Using the formal semantics to understand queries

create table R(a int);
create table S(a int);
create table T(a int);

insert into R values (1);
insert into R values (2);
insert into R values (3);

insert into S values (2);
insert into S values (3);
insert into S values (4);

insert into T values (1);
insert into T values (2);
insert into T values (4);

-- *** what does this query compute ?
select distinct R.a
from R, S
where R.a=S.a;




-- answer: R intersect S

-- *** and this ?
select distinct T.a
from R, S, T
where R.a=T.a and S.a=T.a;
```

```
-- answer: R intersect S


-- *** but what about this one ?
select distinct T.a
from R, S, T
where R.a=T.a or S.a=T.a;







--
-- you might think it is: (R union S) intersect  T
-- but think again! what happens if one of these relations was empty?
-- Let's try it...
delete from R;

select distinct T.a
from R, S, T
where R.a=T.a or S.a=T.a;




-- answer: the query reutnrs  (R union S) intersect  T  if R,S are non-empty
--         otherwise ikt returns the empty set

-------------------------------------------------------------------------------
-- NULLs in SQL

insert into Company(cname, country) values('Apple', 'USA');
insert into Product(pname, price, category, manufacturer) values ('iPad 5', NULL, 'gadget',
'Apple');

-- print nicer:
.nullvalue NULL

select *
from Product;

-- We have a problem now:

select *
from Product
where price < 25;

select *
from Product
where price >= 25;

-- the ipad 5 is nowhere !

-- solution:

select *
from Product
where price is NULL;


-------------------------------------------------------------------------------
-- Complex conditions involving NULL's
--    We need to evaluate in SQL conditions like this:
```

```
--         (price < 25) and (category = 'gadget') or (manufacturer = 'Apple')
--    Suppose price = 19,  category = NULL, and manufacturer = NULL
--    Is the predicate true or false?
--

insert into product(pname,price,category,manufacturer)
   values ('NullProduct', 19.00, null, null);

select *
from product
where (price < 25) and (category = 'gadget') or (manufacturer = 'Apple');


--
--    SQL has 3-valued logic:
--         FALSE = 0       E.g. price<25 is FALSE   when price=99
--         UNKNOWN = 0.5   E.g. price<25 is UNKNOWN when price=NULL
--         TRUE  = 1       E.g. price<25 is TRUE    when price=19
--
--    C1 AND C2    means min(C1,C2)
--    C1 OR  C2    means max(C1,C2)
--    not C        means 1-C
--
--    In class: compute the truth value of the condition above
--
--  The rule for SELECT ... FROM ... WHERE C   is the following:
--      if C = TRUE then include the row in the output
--      if C = FALSE or C = unknown then do not include it

-------------------------------------------------------------------------------
-- Outer joins

insert into Company(cname, country) values ('Google', 'USA');

-- Get everything in the database

select *
from Company x, Product y
where x.cname = y.manufacturer;

-- "Join" is also called an "inner join", and can be written like this:

select *
from Company inner join Product on cname = manufacturer;

-- But we are NOT getting everything in the database !

-- "Left outer join" means: include everything on the left, fill in the right parft  with NULL
values

select *
from Company left outer join Product on cname = manufacturer;
```