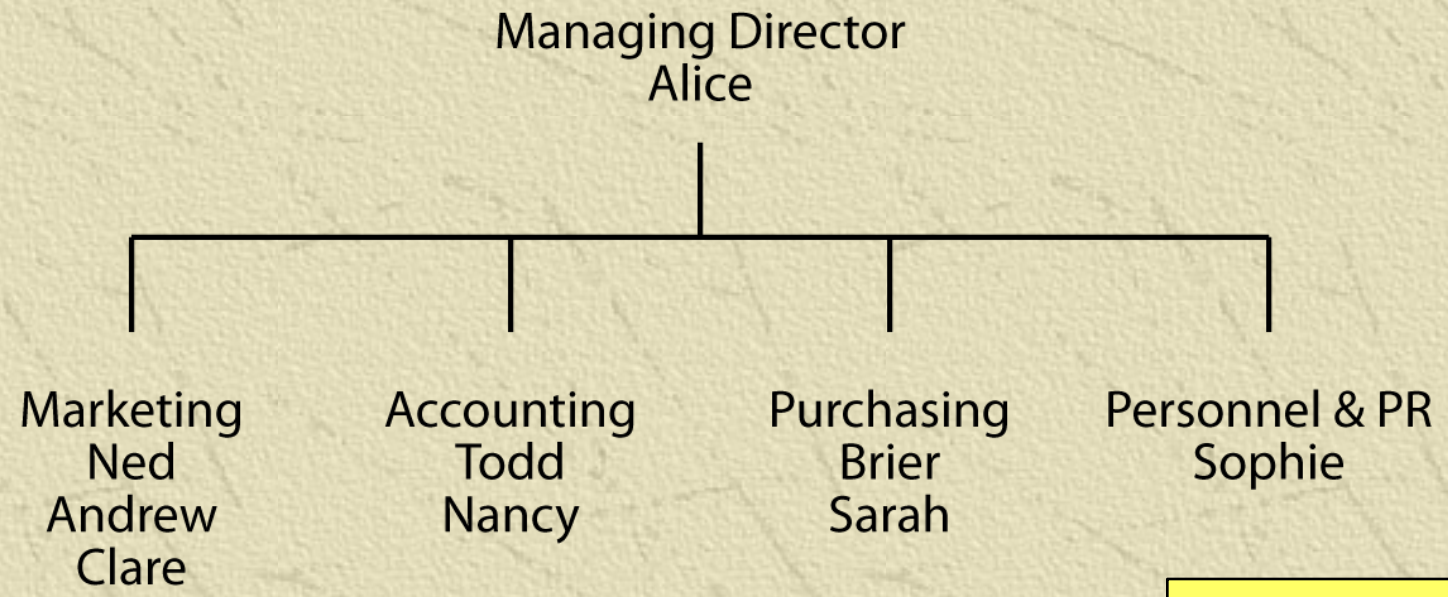
One-to-One and Recursive Relationships

Self-reflection is the school of wisdom

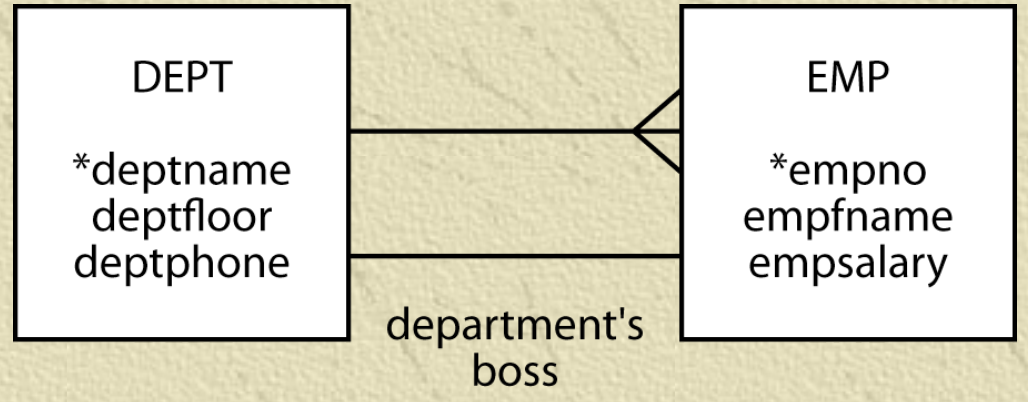
Baltastar Gracián

An organization chart



Every structure for presenting data has an underlying data model

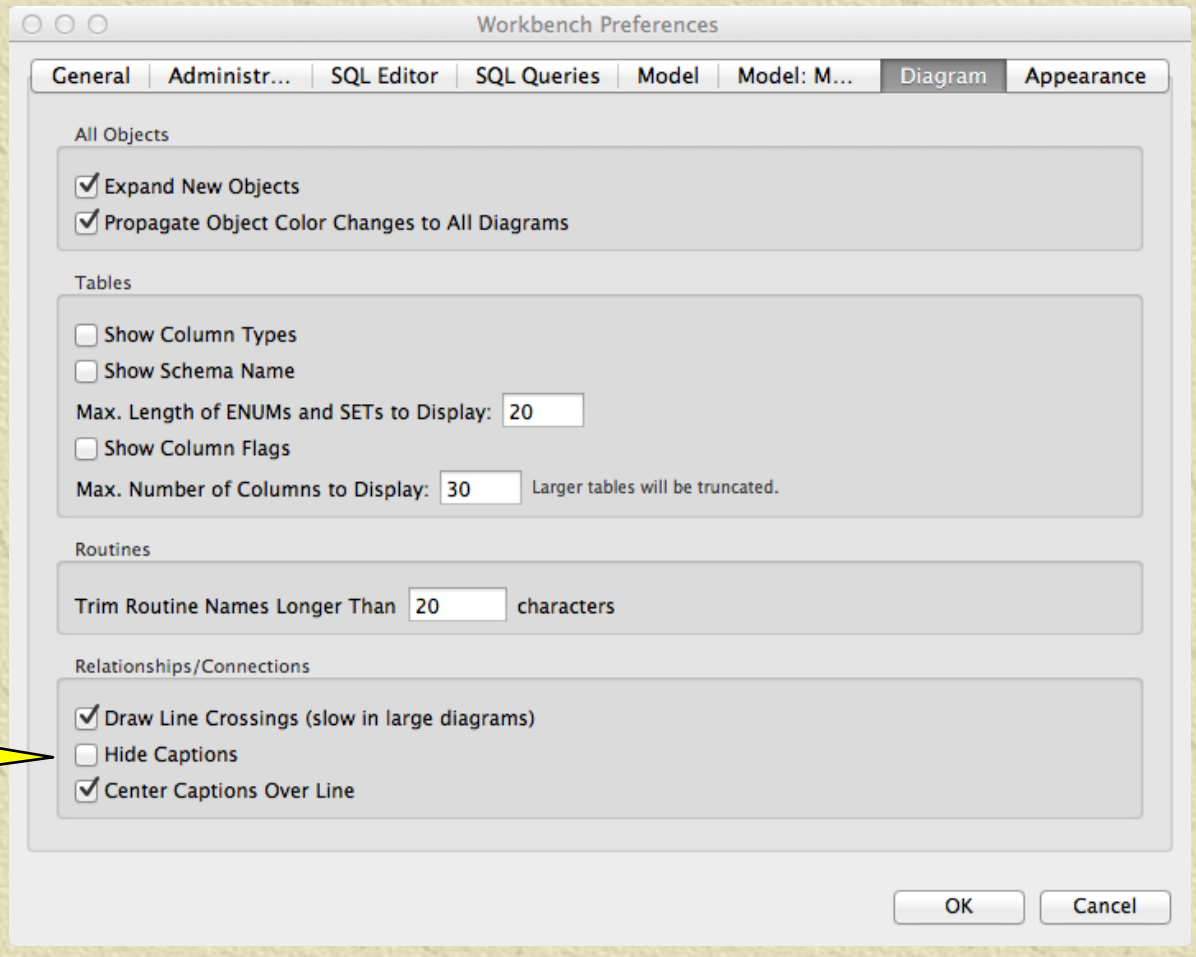
Modeling a 1:1 relationship



- 1:1 relationship is labeled
  - A relationship descriptor
- Obvious relationships are not labeled

Labeling a relationship

Workbench Preferences > Diagram

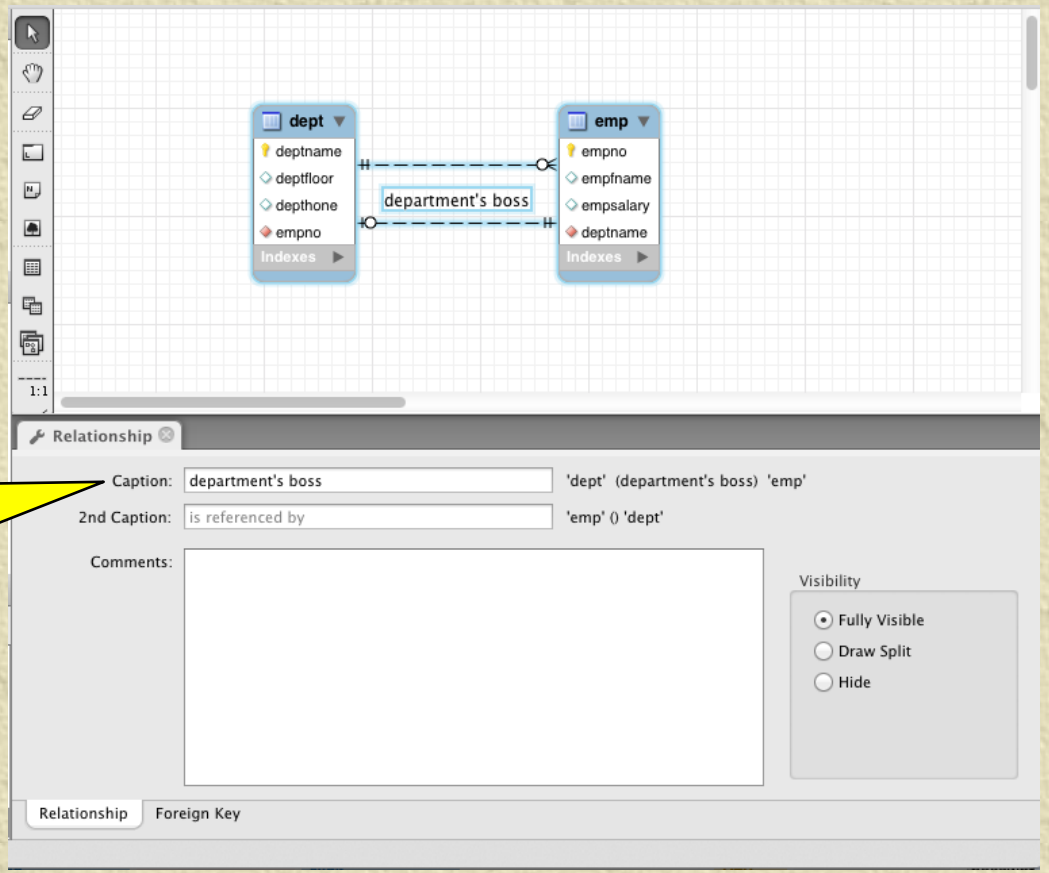


Turn off hide captions

Labeling a relationship

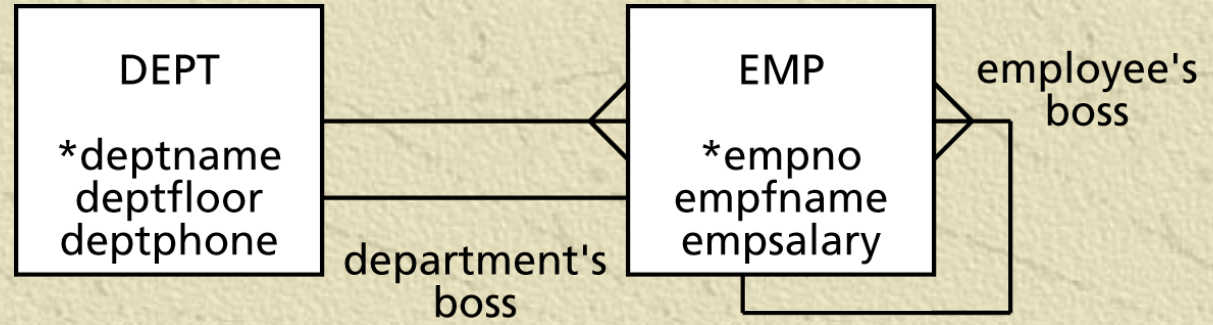
Edit Relationship

Enter caption or blank an unwanted captions

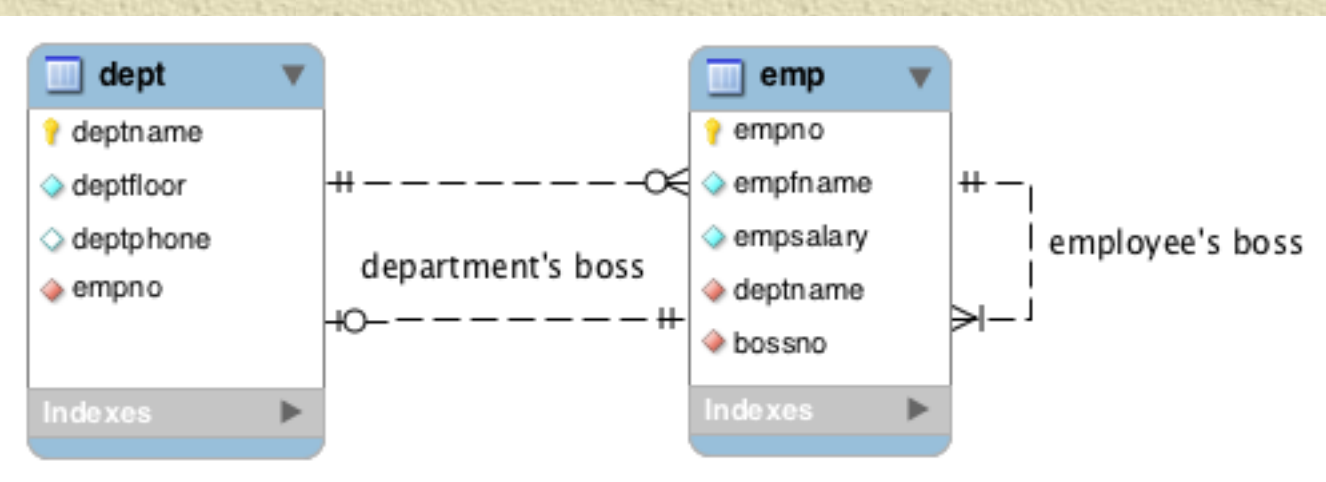


Modeling a recursive relationship

- A recursive relationship relates an entity to itself
- Label recursive relationships



MySQL Workbench



Mapping a 1:1 relationship

- Usual rules apply
- Where do you put the foreign key?
  - `DEPT`
  - `EMP`
  - Both tables

Mapping a recursive relationship

- Usual rules
- 1:m
  - The entity gets an additional column for the foreign key
  - Need a name different from the primary key

Results of the mapping

dept			
deptname	deptfloor	deptphone	*empno*
Management	5	2001	1
Marketing	1	2002	2
Accounting	4	2003	5
Purchasing	4	2004	7
Personnel & PR	1	2005	9

emp				
empno	empfname	empsalary	*deptname*	*bossno*
1	Alice	75000	Management	
2	Ned	45000	Marketing	1
3	Andrew	25000	Marketing	2
4	Clare	22000	Marketing	2
5	Todd	38000	Accounting	1
6	Nancy	22000	Accounting	5
7	Brier	43000	Purchasing	1
8	Sarah	56000	Purchasing	7
9	Sophie	35000	Personnel & PR	1

Creating the tables

```
CREATE TABLE dept (
    deptname    VARCHAR(15),
    deptfloor   SMALLINT   NOT NULL,
    deptphone   SMALLINT   NOT NULL,
    empno       SMALLINT   NOT NULL,
     PRIMARY KEY(deptname));

CREATE TABLE emp (
    empno       SMALLINT,
    empfname    VARCHAR(10),
    empsalary   DECIMAL(7,0),
    deptname    VARCHAR(15),
    bossno      SMALLINT,
     PRIMARY KEY(empno),
     CONSTRAINT fk_belong_dept FOREIGN KEY(deptname)
       REFERENCES dept(deptname),
     CONSTRAINT fk_has_boss foreign key (bossno)
       REFERENCES emp(empno));
```

*There is no foreign key constraint for **empno** in **dept**, because it requires the matching primary key in emp exist. However, the matching foreign key in emp can't be created until the matching primary key in dept exists. **An infinite circle of references.***

Inserting rows

```
INSERT INTO emp (empno, empfname, empsalary, deptname)
   VALUES (1,'Alice',75000,'Management');
INSERT INTO emp VALUES (2,'Ned',45000,'Marketing',1);
INSERT INTO emp VALUES (3,'Andrew',25000,'Marketing',2);
INSERT INTO emp VALUES (4,'Clare',22000,'Marketing',2);
INSERT INTO emp VALUES (5,'Todd',38000,'Accounting',1);
INSERT INTO emp VALUES (6,'Nancy',22000,'Accounting',5);
INSERT INTO emp VALUES (7,'Brier',43000,'Purchasing',1);
INSERT INTO emp VALUES (8,'Sarah',56000,'Purchasing',7);
INSERT INTO emp VALUES (9,'Sophie',35000,'Personnel',1);
```

Order the INSERTs to avoid referential integrity problems

Exercise

- Several Olympic events are team sports (e.g., basketball, relays) and some involve a pair of athletes (e.g., kayaking, rowing, beach volleyball)
- A team can have a captain
- A country has a flag bearer
- There can be some husband and wife pairs at a games (e.g., Jared Tallent and Claire Woods from Australia)
- Draw a data model to record these details

Querying a 1:1 relationship

List the salary of each department's boss.

```
SELECT empfname, deptname, empsalary FROM emp
   WHERE empno IN (SELECT empno FROM dept);
```

empfname	deptname	empsalary
Alice	Management	75000
Ned	Marketing	45000
Todd	Accounting	38000
Brier	Purchasing	43000
Sophie	Personnel & PR	35000

Querying a 1:1 relationship

List the salary of each department's boss.

```
SELECT empfname, dept.deptname, empsalary FROM emp, dept
   WHERE dept.empno = emp.empno;
```

empfname	deptname	empsalary
Alice	Management	75000
Ned	Marketing	45000
Todd	Accounting	38000
Brier	Purchasing	43000
Sophie	Personnel & PR	35000

Joining a table with itself

Find the salary of Nancy's boss.

wrk

empno	empfname	empsalary	*deptname*	*bossno*
1	Alice	75000	Management	
2	Ned	45000	Marketing	1
3	Andrew	25000	Marketing	2
4	Clare	22000	Marketing	2
5	Todd	38000	Accounting	1
6	Nancy	22000	Accounting	5
7	Brier	43000	Purchasing	1
8	Sarah	56000	Purchasing	7
9	Sophie	35000	Personnel & PR	1

boss

empno	empfname	empsalary	*deptname*	*bossno*
1	Alice	75000	Management	
2	Ned	45000	Marketing	1
3	Andrew	25000	Marketing	2
4	Clare	22000	Marketing	2
5	Todd	38000	Accounting	1
6	Nancy	22000	Accounting	5
7	Brier	43000	Purchasing	1
8	Sarah	56000	Purchasing	7
9	Sophie	35000	Personnel & PR	1

Querying a recursive relationship

Find the salary of Nancy's boss.

```
SELECT wrk.empfname, wrk.empsalary, boss.empfname, boss.empsalary
    FROM emp wrk, emp boss
        WHERE wrk.empfname = 'Nancy'
        AND wrk.bossno = boss.empno;
```

wrk					boss				
empno	empfname	empsalary	deptname	bossno	empno	empfname	empsalary	deptname	bossno
2	Ned	45,000	Marketing	1	1	Alice	75,000	Management	
3	Andrew	25,000	Marketing	2	2	Ned	45,000	Marketing	1
4	Clare	22,000	Marketing	2	2	Ned	45,000	Marketing	1
5	Todd	38,000	Accounting	1	1	Alice	75,000	Management	
6	Nancy	22,000	Accounting	5	5	Todd	38,000	Accounting	1
7	Brier	43,000	Purchasing	1	1	Alice	75,000	Management	
8	Sarah	56,000	Purchasing	7	7	Brier	43,000	Purchasing	1
9	Sophie	35,000	Personnel & PR	1	1	Alice	75,000	Management	

wrk.empfname	wrk.empsalary	boss.empfname	boss.empsalary
Nancy	22000	Todd	3800

Querying a recursive relationship

Find the names of employees who earn more than their boss.

```
SELECT wrk.empfname
    FROM emp wrk, emp boss
        WHERE wrk.bossno = boss.empno
          AND wrk.empsalary > boss.empsalary;
```

wrk					boss				
empno	empfname	empsalary	deptname	bossno	empno	empfname	empsalary	deptname	bossno
2	Ned	45,000	Marketing	1	1	Alice	75,000	Management	
3	Andrew	25,000	Marketing	2	2	Ned	45,000	Marketing	1
4	Clare	22,000	Marketing	2	2	Ned	45,000	Marketing	1
5	Todd	38,000	Accounting	1	1	Alice	75,000	Management	
6	Nancy	22,000	Accounting	5	5	Todd	38,000	Accounting	1
7	Brier	43,000	Purchasing	1	1	Alice	75,000	Management	
8	Sarah	56,000	Purchasing	7	7	Brier	43,000	Purchasing	1
9	Sophie	35,000	Personnel & PR	1	1	Alice	75,000	Management	

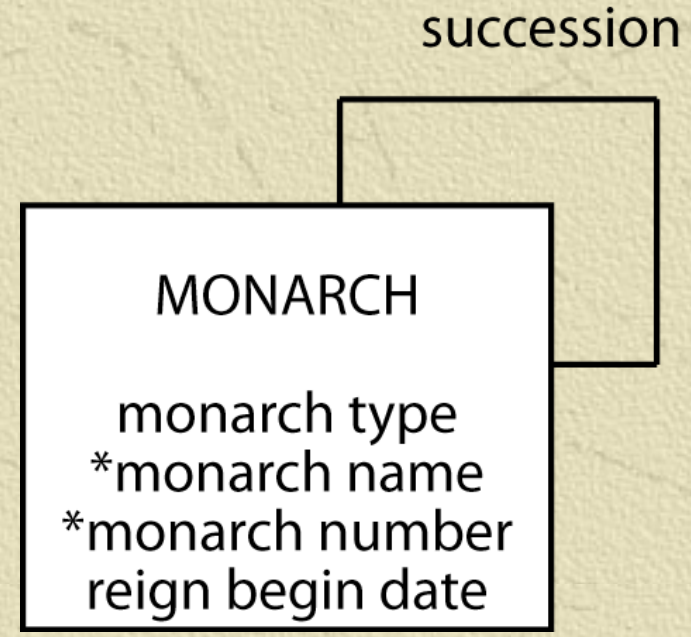
empfname
Sarah

Exercise

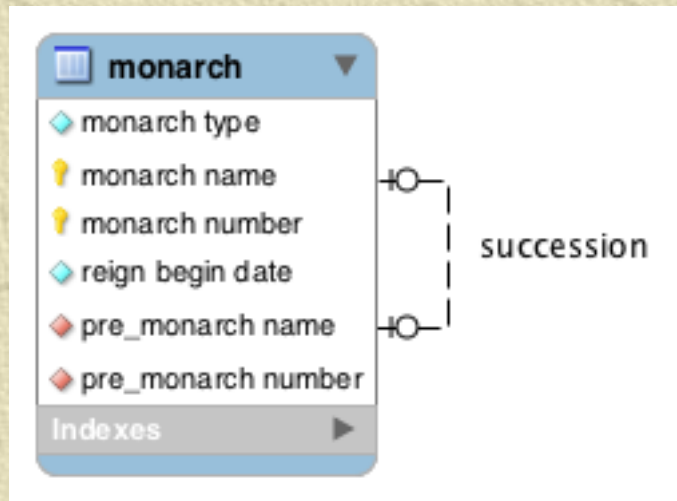
- Find the names of employees in the same department as their boss

Modeling a 1:1 recursive relationship

- The English monarchy



MySQL Workbench



Mapping a 1:1 recursive relationship

monarch					
montype	monname	monnum	rgnbeg	*premonname*	*premonnum*
Queen	Victoria	I	1837/6/20	William	IV
King	Edward	VII	1901/1/22	Victoria	I
King	George	V	1910/5/6	Edward	VII
King	Edward	VIII	1936/1/20	George	V
King	George	VI	1936/12/11	Edward	VIII
Queen	Elizabeth	II	1952/2/6	George	VI

Creating the table

```
CREATE TABLE monarch (
   montype       VARCHAR(5),
   monname       VARCHAR(15) NOT NULL,
   monnum        VARCHAR(5)  NOT NULL,
   rgnbeg        DATE,
   premonname    VARCHAR(15),
   premonnum     VARCHAR(5),
      PRIMARY KEY(monname,monnum),
      CONSTRAINT fk_monarch
      FOREIGN KEY (premonname, premonnum)
      REFERENCES monarch(monname, monnum));
```

Inserting rows

```
INSERT INTO monarch (montype,monname, monnum,rgnbeg)
   VALUES ('King','William','IV','1830-06-26');
INSERT INTO monarch
    VALUES ('Queen','Victoria','I','1837-06-20','William','IV');
INSERT INTO monarch
    VALUES ('King','Edward','VII','1901-01-22','Victoria','I');
INSERT INTO monarch
    VALUES ('King','George','V','1910-05-06','Edward','VII');
INSERT INTO monarch
    VALUES ('King','Edward','VIII','1936-01-20','George','V');
INSERT INTO monarch
    VALUES ('King','George','VI','1936-12-11','Edward','VIII');
INSERT INTO monarch
    VALUES 'Queen','Elizabeth','II','1952-02-06','George','VI');
```

Exercise

- Design a database to record details of all Olympic cities
  - Recognize that a city can host an Olympics more than once, though a particular Olympics is in only one city at a time
  - Recognize that each Olympics has only one predecessor and successor

Querying a 1:1 recursive relationship

Who preceded Elizabeth II?

```
SELECT premonname, premonnum FROM monarch
   WHERE monname = 'Elizabeth' and monnum = 'II';
```

premonname	premonnum
George	VI

Querying a 1:1 recursive relationship

Was Elizabeth II's predecessor a king or queen?

```
SELECT pre.montype FROM monarch cur, monarch pre
   WHERE cur.premonname = pre.monname
   AND cur.premonnum = pre.monnum
   AND cur.monname = 'Elizabeth'
   AND cur.monnum = 'II';
```

montype
King

cur						pre					
montype	monname	monnum	rgnbeg	*premonname*	*premonnum*	montype	monname	monnum	rgnbeg	*premonname*	*premonnum*
Queen	Elizabeth	II	1952/2/6	George	VI	King	George	VI	1936/12/11	Edward	VIII

Querying a 1:1 recursive relationship

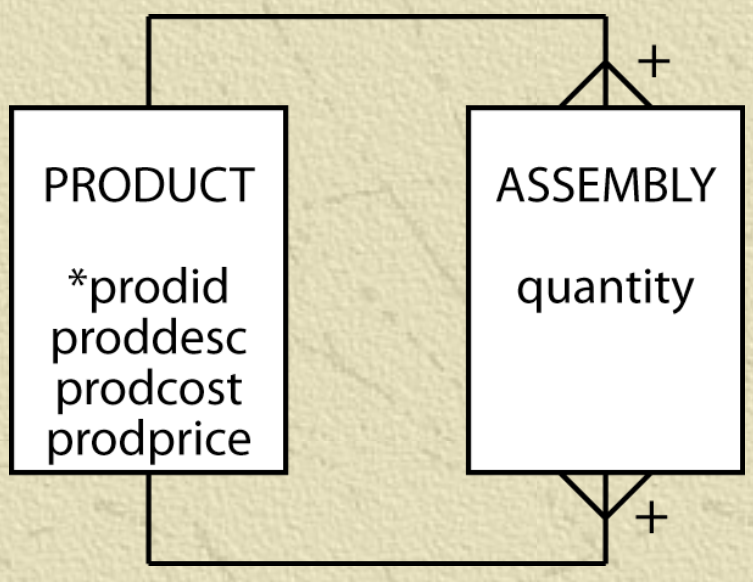
List the kings and queens of England in ascending chronological order.

```
SELECT montype, monname, monnum, rgnbeg
   FROM monarch ORDER BY rgnbeg;
```

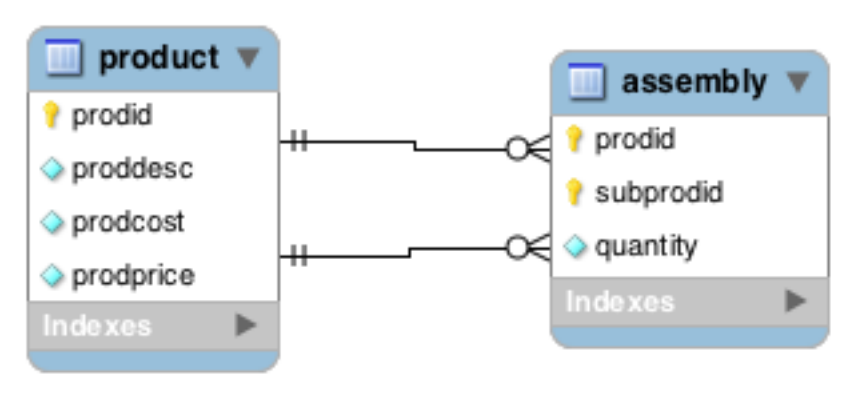
montype	monname	monnum	rgnbeg
Queen	Victoria	I	1837-06-20
King	Edward	VII	1901-01-22
King	George	V	1910-05-06
King	Edward	VIII	1936-01-20
King	George	VI	1936-12-11
Queen	Elizabeth	II	1952-02-06

Modeling an m:m recursive relationship

- Bill of materials problem
- A product can appear as part of many other products and can be made up of many products



Modeling an m:m recursive relationship



Mapping an m:m recursive relationship

product			
prodid	proddesc	prodcost	prodprice
1000	Animal photography kit		725
101	Camera	150	300
102	Camera case	10	15
103	70-210 zoom lens	125	200
104	28-85 zoom lens	115	185
105	Photographer's vest	25	40
106	Lens cleaning cloth	1	1.25
107	Tripod	35	45
108	16 GB SDHC memory card	30	30

assembly		
quantity	*prodid*	*subprodid*
1	1000	101
1	1000	102
1	1000	103
1	1000	104
1	1000	105
2	1000	106
1	1000	107
4	1000	108

Creating the tables

```
CREATE TABLE product (
   prodid      INTEGER,
   proddesc    VARCHAR(30),
   prodcost    DECIMAL(9,2),
   prodprice   DECIMAL(9,2),
      PRIMARY KEY(prodid));

CREATE TABLE assembly (
   quantity    INTEGER NOT NULL,
   prodid      INTEGER,
   subprodid   INTEGER,
      PRIMARY KEY(prodid, subprodid),
      CONSTRAINT fk_assembly_product FOREIGN KEY(prodid)
         REFERENCES product(prodid),
      CONSTRAINT fk_assembly_subproduct FOREIGN KEY(subprodid)
         REFERENCES product (prodid));
```

Exercise

- In a round-robin tournament, each contestant meets all other contestants in turn
- In the Olympics, it is common for an event with a large pool of contestants to be broken into groups, with a round-robin tournament in each group to determine who advances from the group to the next level
- Design a data model to record details of a round-robin competition

Querying an m:m recursive relationship

List the product identifier of each component of the animal photography kit.

```
SELECT subprodid FROM product, assembly
   WHERE proddesc = 'Animal photography kit'
   AND product.prodid = assembly.prodid;
```

subprodid
101
106
107
105
104
103
102
108

Querying an m:m recursive relationship

List the product description and cost of each component of the animal photography kit.

```
SELECT proddesc, prodcost FROM product
   WHERE prodid IN
     (SELECT subprodid FROM product, assembly
        WHERE proddesc = 'Animal photography kit'
        AND product.prodid = assembly.prodid);
```

proddesc	prodcost
Camera	150.00
Camera case	10.00
70-210 zoom lens	125.00
28-85 zoom lens	115.00
Photographer's vest	25.00
Lens cleaning cloth	1.00
Tripod	35.00
16 GB SDHC memory card	30.00

Exercises

- Model the following situations
  - Friendship
  - Course prerequisites
  - A matrix organization where a person can report to multiple people

Conclusion

- Introduced
  - Recursive relationship
  - Self-referential constraint
  - Self-join