

Data Modeling and Databases

Lab 6: Recitation

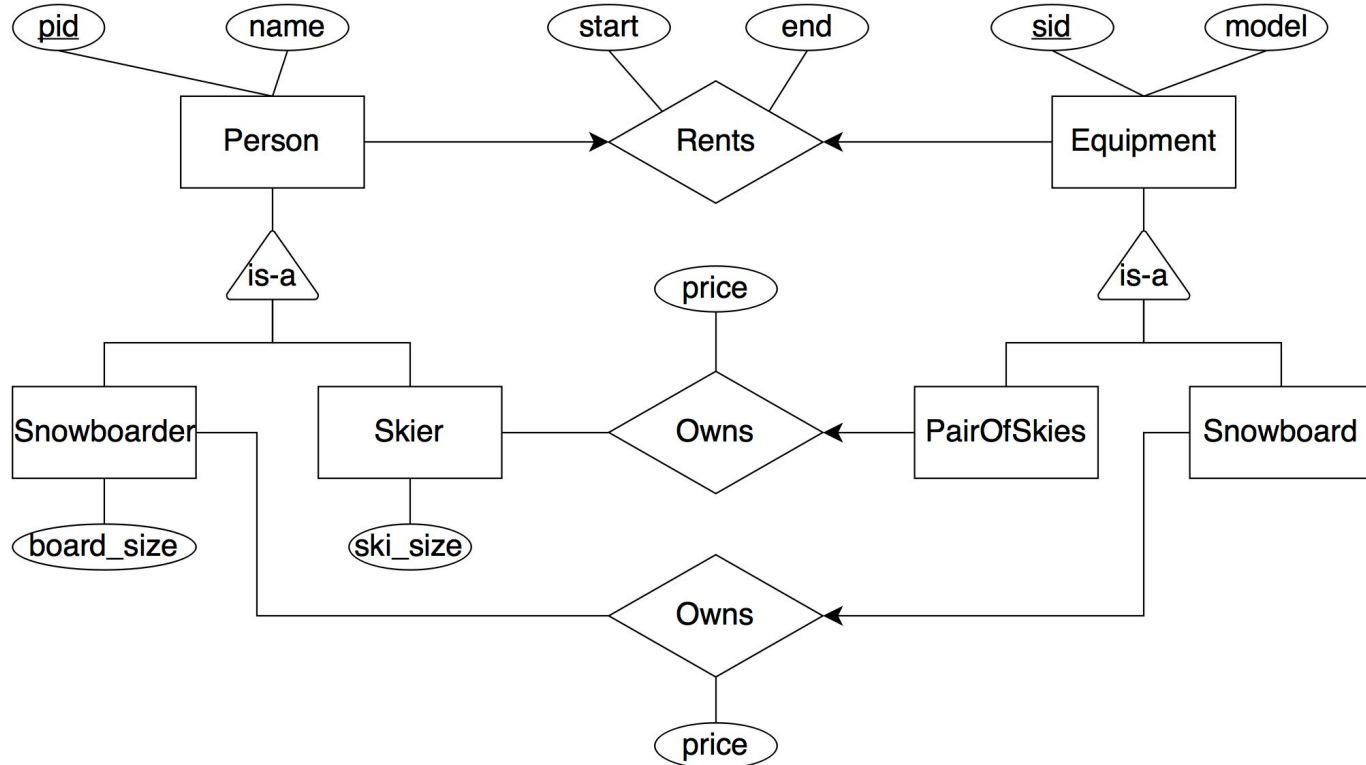
Bulat Gabbasov, Albina Khusainova
Innopolis University
2016

Q1

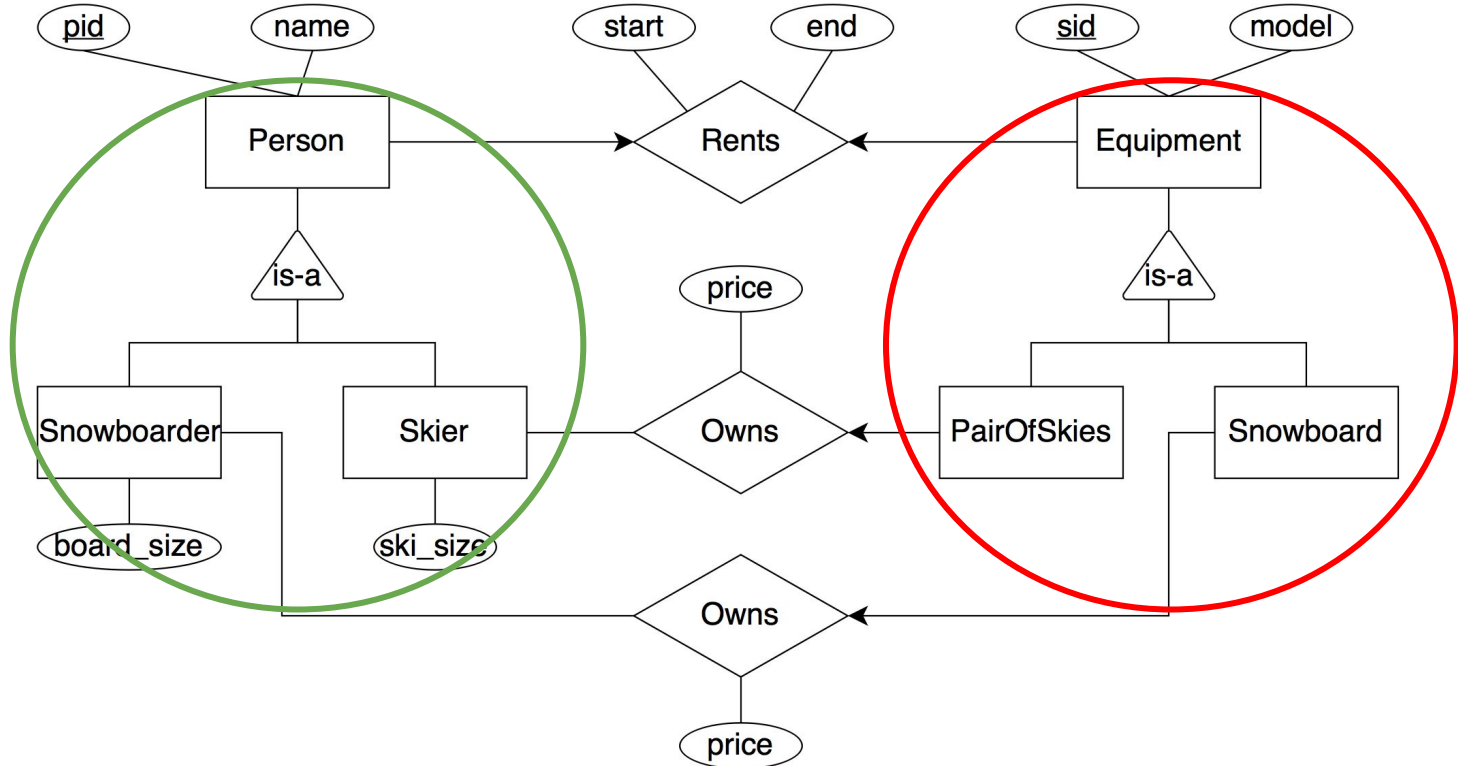
(10 points) **Design an E/R diagram describing the following domain:**

- A Person has attributes pid (key) and name.
- A Skier is a type of Person with attribute ski_size.
- A Snowboarder is a type of Person with attribute board_size.
- A PairOfSkis has attribute sid (key) and model.
- A Snowboard has attribute sid (key) and model.
- A Skier owns zero or more PairOfSkis. The ownership relation has a purchase price. A PairOfSkis is owned by at most one Skier.
- A Snowboarder owns zero or more Snowboards. The ownership relation has a purchase price. A Snowboard is owned by at most one Snowboarder.
- A Person can rent a PairOfSkis or a Snowboard. A person cannot rent more than one PairOfSkis or one Snowboard at the same time. A person cannot rent a PairOfSkis and a Snowboard at the same time either. A piece of equipment can be rented by at most one person at a time. The rental comes with a start date and an end date.

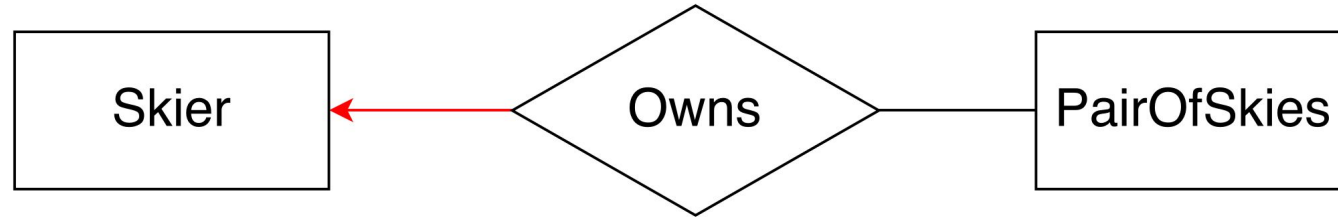
Q1: Solution



Q1: Solution



Q1. Common mistakes: Owns relationship

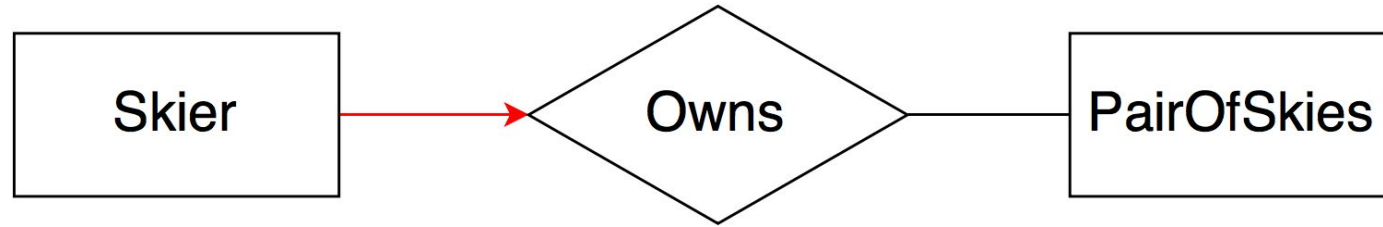


A Skier owns **zero or more** PairOfSkis. The ownership relation has a purchase price.

A PairOfSkis is **owned by at most one** Skier.

Messing up the notation

Q1. Common mistakes: Owns relationship

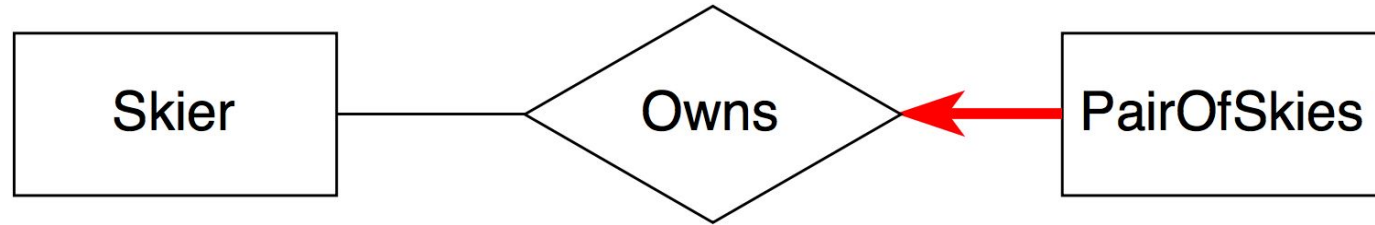


A Skier owns **zero or more** PairOfSkis. The ownership relation has a purchase price.

A PairOfSkis is **owned by at most one** Skier.

Only one pair of skies for a Skier?!

Q1. Common mistakes: Owns relationship

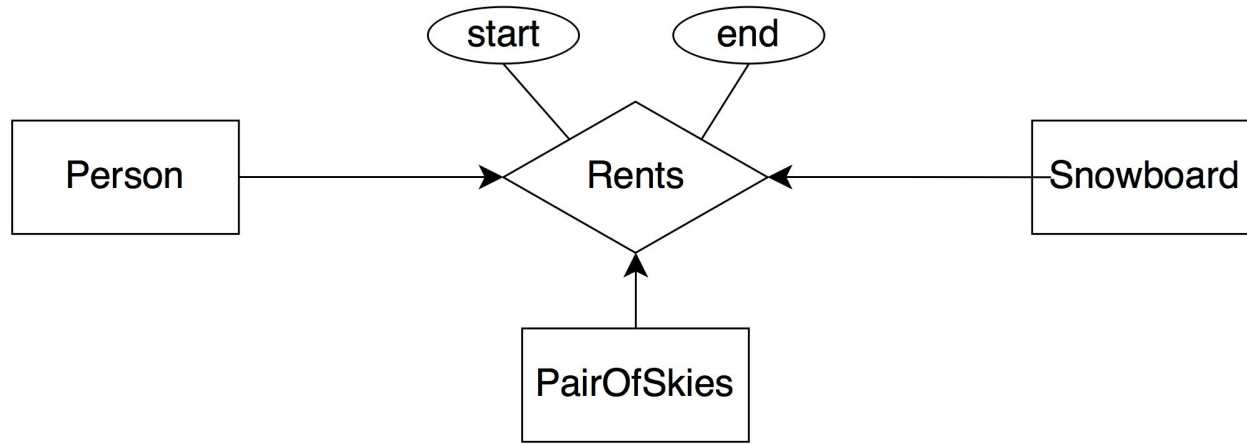


A Skier owns **zero or more** PairOfSkis. The ownership relation has a purchase price.

A PairOfSkis is **owned by at most one** Skier.

Should each and every pair of skies be owned by someone?!

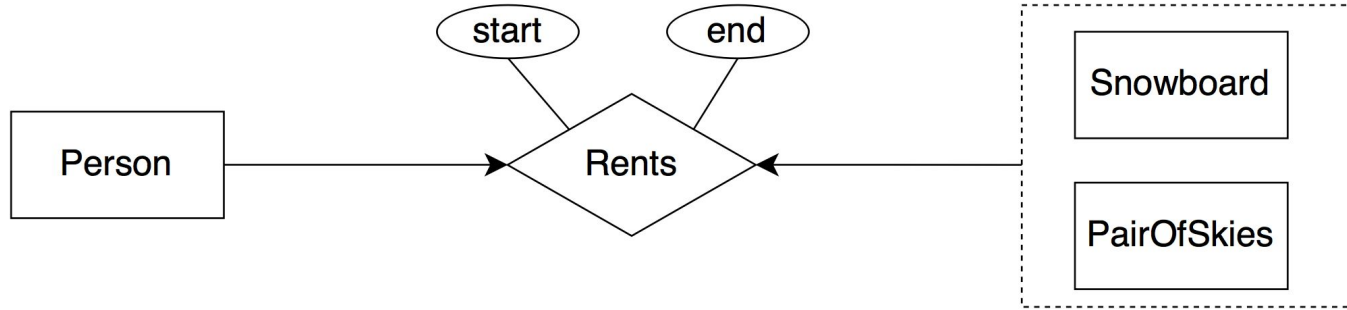
Q1. Common mistakes: Rents relationship



A Person can rent a PairOfSkis **or** a Snowboard. A person **cannot rent more than one** PairOfSkis or one Snowboard **at the same time**. A person **cannot rent** a PairOfSkis and a Snowboard **at the same time either**. A piece of equipment **can be rented by at most one person** at a time. The rental comes with a start date and an end date.

Why is this not right?

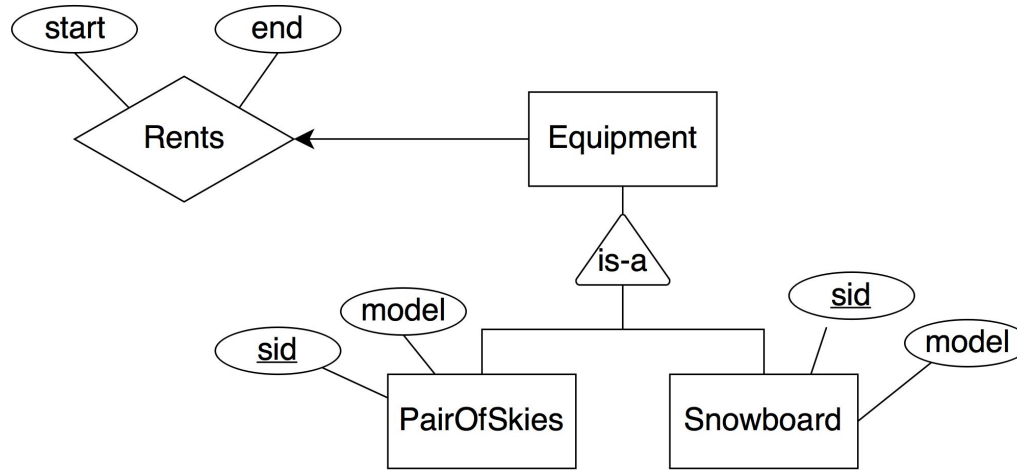
Q1. Common mistakes: Rents relationship



A Person can rent a PairOfSkis **or** a Snowboard. A person **cannot rent more than one** PairOfSkis or one Snowboard **at the same time**. A person **cannot rent** a PairOfSkis and a Snowboard **at the same time either**. A piece of equipment **can be rented by at most one person** at a time. The rental comes with a start date and an end date.

Why is this not right?

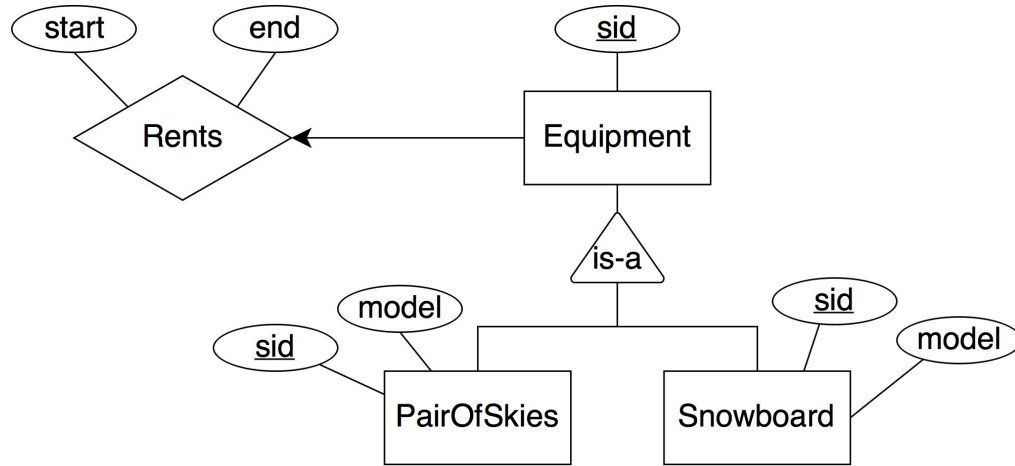
Q1. Common mistakes: Rents relationship



A Person can rent a PairOfSkis **or** a Snowboard. A person **cannot rent more than one** PairOfSkis or one Snowboard **at the same time**. A person **cannot rent** a PairOfSkis and a Snowboard **at the same time either**. A piece of equipment **can be rented by at most one person** at a time. The rental comes with a start date and an end date.

Why is this not right?

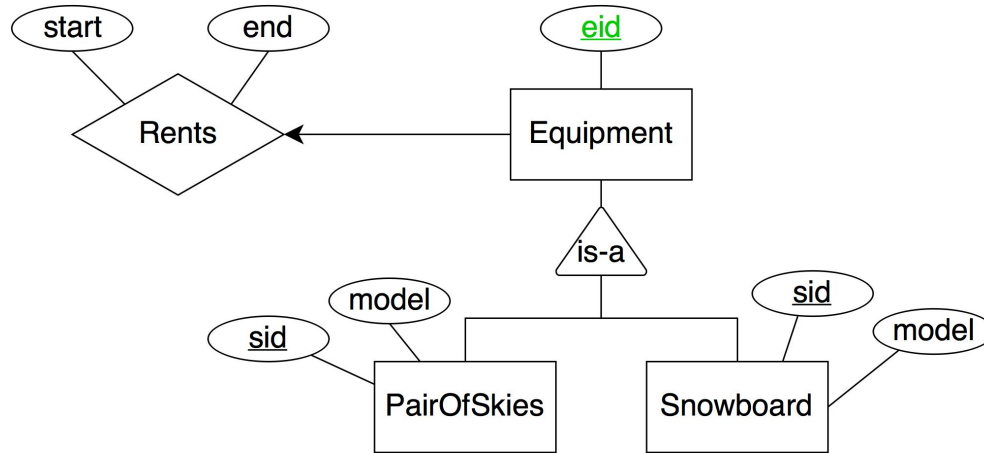
Q1. Common mistakes: Rents relationship



A Person can rent a PairOfSkis **or** a Snowboard. A person **cannot rent more than one** PairOfSkis or one Snowboard **at the same time**. A person **cannot rent** a PairOfSkis and a Snowboard **at the same time either**. A piece of equipment **can be rented by at most one person** at a time. The rental comes with a start date and an end date.

Duplicate sid fields

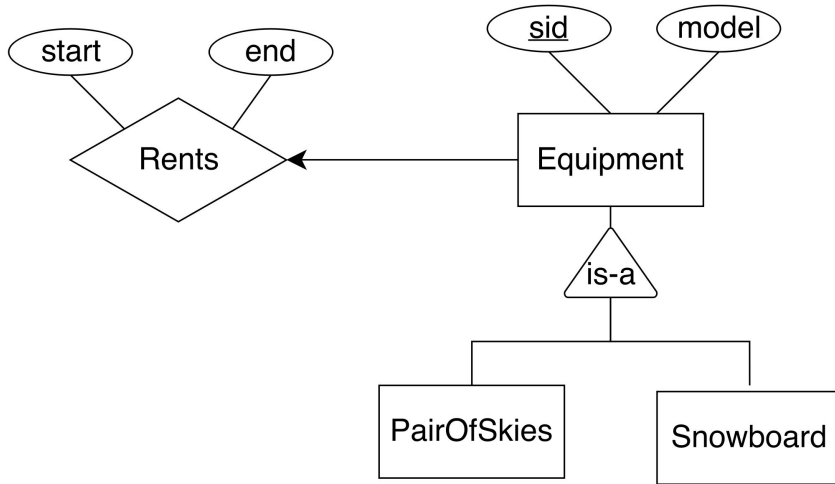
Q1. Common mistakes: Rents relationship



A Person can rent a PairOfSkis **or** a Snowboard. A person **cannot rent more than one** PairOfSkis or one Snowboard **at the same time**. A person **cannot rent** a PairOfSkis and a Snowboard **at the same time either**. A piece of equipment **can be rented by at most one person** at a time. The rental comes with a start date and an end date.

Possible, but why have two ids?

Q1. Common mistakes: Rents relationship

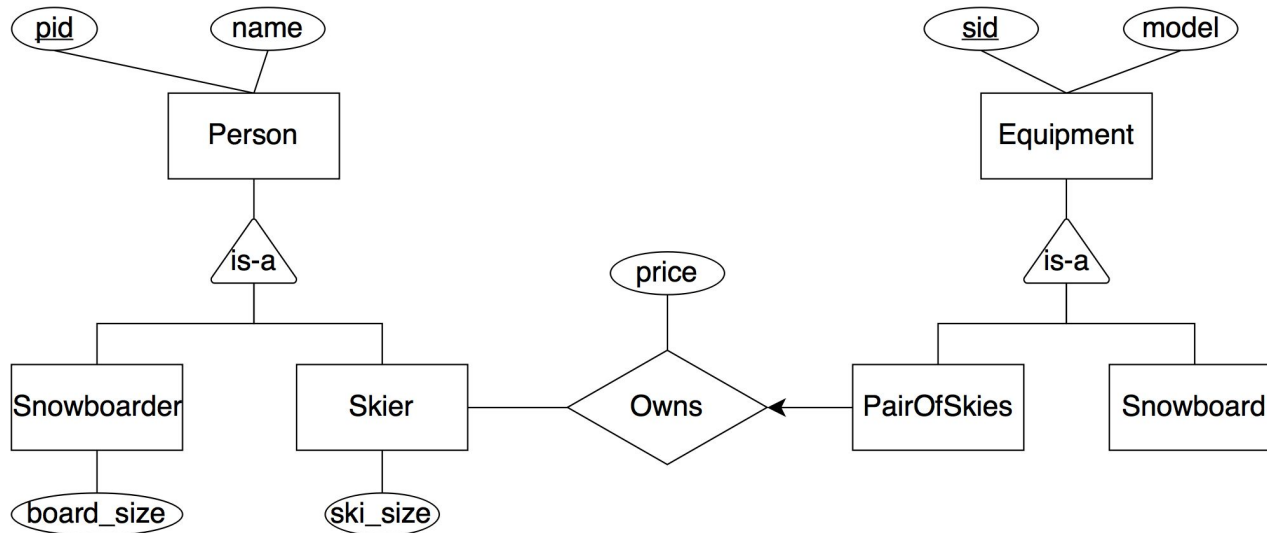


A Person can rent a PairOfSkis **or** a Snowboard. A person **cannot rent more than one** PairOfSkis or one Snowboard **at the same time**. A person **cannot rent** a PairOfSkis and a Snowboard **at the same time either**. A piece of equipment **can be rented by at most one person** at a time. The rental comes with a start date and an end date.

A better option

Q2

(6 points) Write the SQL CREATE TABLE statement for the owns relation between Skier and PairOfSkis. Make sure that your statement specifies the PRIMARY KEY and any FOREIGN KEYS. Additionally, we would like to enforce the constraint that purchase price be greater than zero.



Q2: Solution

```
CREATE TABLE owns (  
    sid INT PairOfSkis,  
    pid INT Skier,  
    purchase_price INT,  
    PRIMARY KEY (sid),  
    FOREIGN KEY (sid) REFERENCES PairOfSkis,  
    FOREIGN KEY (pid) REFERENCES Skier,  
    CHECK ( purchase_price > 0)  
)
```

Q2. Common mistakes: PK choice

```
CREATE TABLE owns (  
    sid INT PairOfSkis,  
    pid INT Skier,  
    purchase_price INT,  
    PRIMARY KEY (sid,pid) (sid),  
    FOREIGN KEY (sid) REFERENCES PairOfSkis,  
    FOREIGN KEY (pid) REFERENCES Skier,  
    CHECK ( purchase_price > 0)  
)
```


Q2. Common mistakes: PK choice

```
CREATE TABLE owns (  
    sid INT PairOfSkis,  
    pid INT Skier,  
    purchase_price INT,  
    PRIMARY KEY (sid,pid) (sid),  
    FOREIGN KEY (sid) REFERENCES PairOfSkis,  
    FOREIGN KEY (pid) REFERENCES Skier,  
    CHECK ( purchase_price > 0)  
)
```

pid	sid
skierA	ski5
skierB	ski5

“A PairOfSkis is owned by at most one Skier.”

Q2. Common mistakes: PK choice

```
CREATE TABLE owns (  
    sid INT PairOfSkis,  
    pid INT Skier,  
    purchase_price INT,  
    PRIMARY KEY (pid) (sid),  
    FOREIGN KEY (sid) REFERENCES PairOfSkis,  
    FOREIGN KEY (pid) REFERENCES Skier,  
    CHECK ( purchase_price > 0)  
)
```

Q2. Common mistakes: PK choice

```
CREATE TABLE owns (  
  sid INT PairOfSkis,  
  pid INT Skier,  
  purchase_price INT,  
  PRIMARY KEY (pid) (sid),  
  FOREIGN KEY (sid) REFERENCES PairOfSkis,  
  FOREIGN KEY (pid) REFERENCES Skier,  
  CHECK ( purchase_price > 0)  
)
```

pid	sid
skierA	ski5
skierA	ski6

Only one pair of skis for one skier?! “A Skier owns **zero or more** PairOfSkis.”

Q2. Common mistakes: Excessive attributes

```
CREATE TABLE owns (  
    sid INT PairOfSkis,  
    pid INT Skier,  
    purchase_price INT,  
    model varchar,  
    ski_size INT  
    PRIMARY KEY (sid),  
    FOREIGN KEY (sid) REFERENCES PairOfSkis,  
    FOREIGN KEY (pid) REFERENCES Skier,  
    FOREIGN KEY (model) REFERENCES PairOfSkis,  
    FOREIGN KEY (ski_size) REFERENCES Skier,  
    CHECK ( purchase_price > 0)  
)
```

What for?!

Q2. Second option - combining Owns and PairOfSkis

```
CREATE TABLE pairOfSkisOwns (  
    sid INT PairOfSkis,  
    model VARCHAR,  
    pid INT Skier,  
    purchase_price INT,  
    PRIMARY KEY (sid),  
    FOREIGN KEY (pid) REFERENCES Skier,  
    CHECK ( purchase_price > 0)  
)
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

```
SELECT firstName, lastName from Driver d1

WHERE EXISTS (SELECT 1 FROM Driver d2

              WHERE d1.lastname = d2.lastname

              AND d1.firstName != d2.firstname)
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

Forgetting to remove self references

```
SELECT firstName, lastName from Driver d1  
  
WHERE EXISTS (SELECT 1 FROM Driver d2  
  
              WHERE d1.lastname = d2.lastname)
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

Comparing lastName with a set that possibly has multiple elements

```
SELECT firstName, lastName from Driver d1

WHERE lastName = (SELECT lastName FROM Driver d2

                  WHERE d1.lastName = d2.lastName

                  AND d1.firstName != d2.firstName)
```


Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

Use IN instead

```
SELECT firstName, lastName from Driver d1
WHERE lastName IN (SELECT lastName FROM Driver d2
                  WHERE d1.lastName = d2.lastName
                  AND d1.fistName != d2.firstName)
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

```
SELECT DISTINCT d1.firstName, d1.lastName from Driver d1, Driver d2
WHERE d1.lastName = d2.lastName AND d1.firstName != d2.firstName
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

Forgetting to put distinct

```
SELECT DISTINCT d1.firstName, d1.lastName from Driver d1, Driver d2  
  
WHERE d1.lastName = d2.lastName AND d1.firstName != d2.firstName
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

Using `<` instead of `!=`

```
SELECT DISTINCT d1.firstName, d1.lastName from Driver d1, Driver d2
WHERE d1.lastName = d2.lastName AND d1.firstName < d2.firstName
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.1 Write a query in SQL to give the first and last names of all drivers that share a last name with another driver.

```
SELECT d1.firstName, d1.lastName from Driver d1
WHERE d1.lastName IN (SELECT d2.lastName FROM Driver d2
                      GROUP BY d2.lastName
                      HAVING COUNT(firstName) > 1)
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.2 (5 points) Write a query in SQL to find all people (first name, last name) who are both voters from district '32' and drivers under the age 25.

Almost no issues with this question.

```
SELECT firstName, lastName FROM Driver WHERE age < 25
```

```
INTERSECT
```

```
SELECT firstName, lastName FROM Voter WHERE district = '32'
```

Q3: Consider the two tables:

Table Driver (licenseNum, firstName, lastName, age) – part of a simple driver registration database. Every row of Driver has a unique licenceNum.

Also consider table Voter (voterID, firstName, lastName, district) – where every row of Voter has a unique voterID.

3.2 (5 points) Write a query in SQL to find all people (first name, last name) who are both voters from district '32' and drivers under the age 25.

Almost no issues with this question.

```
SELECT firstName, lastName
FROM Driver NATURAL JOIN Voter
WHERE age < 25 AND district = '32'
```

Q4: Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Find the names of suppliers who supply some red part.

$$\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog) \bowtie Supplier)$$

```
SELECT DISTINCT S.name
```

```
FROM Parts P, Catalog C, Supplier S
```

```
WHERE P.color = 'red' AND P.pid = C.pid AND C.sid = S.sid
```


Q4: Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Find the names of suppliers who supply some red part.

$$\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog) \bowtie Supplier)$$

```
SELECT DISTINCT S.name
```

```
FROM Parts P, Catalog C, Supplier S
```

```
WHERE P.color = 'red' AND P.pid = C.pid AND C.sid = S.sid
```

Q4: Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Find the names of suppliers who supply some red part.

$$\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog) \bowtie Supplier)$$

```
SELECT S.name
```

```
FROM Supplier s WHERE s.sid IN (SELECT c.sid FROM Catalog c
```

```
WHERE c.pid IN (SELECT p.pid FROM Parts p WHERE p.color = 'red'))
```

Q4: Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Find the sids of suppliers who supply some red or green part.

$$\pi_{sid}(\pi_{pid}(\sigma_{color='red' \vee color='green'} Parts) \bowtie Catalog)$$

```
SELECT C.sid
```

```
FROM Parts P, Catalog C
```

```
WHERE (P.color = 'red' OR P.color = 'green') AND P.pid = C.pid
```

Q4: Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Find the sids of suppliers who supply some red or green part.

$$\pi_{sid} \left(\left(\pi_{pid}(\sigma_{color='red'} Parts) \cup \pi_{pid}(\sigma_{color='green'} Parts) \right) \bowtie Catalog \right)$$

```
SELECT C.sid
```

```
FROM (SELECT p.pid FROM Parts P WHERE P.color = 'red'
```

```
UNION SELECT p.pid FROM Parts P WHERE P.color = 'green') PS, Catalog C
```

```
WHERE PS.pid = C.pid
```

Q5

5.1 Which of the following relational algebra operations do not require the participating tables to be union-compatible?

- (A) Union
- (B) Intersection
- (C) Difference
- (D) Join**

5.2 Relational Algebra does not have

- (A) Selection operator.
- (B) Projection operator.
- (C) Aggregation operators.**
- (D) Division operator.

5.3 In an E-R diagram a thick line indicate

- (A) Total participation.**
- (B) Multiple participation.
- (C) Cardinality N.
- (D) None of the above.

Q5

5.4 The operation which is not considered a basic operation of relational algebra is

- (A) Join.
- (B) Selection.
- (C) Union.
- (D) Cross product.

5.5 In SQL the statement `select * from R, S` is equivalent to

- (A) `select * from R natural join S`.
- (B) `select * from R cross join S`. (cross product)
- (C) `(select * from R) union (select * from S)`.
- (D) `(select * from R) intersect (select * from S)`.

5.6 In SQL, testing whether a subquery is empty is done using

- (A) DISTINCT
- (B) UNIQUE
- (C) NULL
- (D) EXISTS

Q5

5.7 A trigger is?

- (A) A statement that is executed automatically by the system as a side effect of modification to the
- (B) A statement that enables to start any DBMS
- (C) A statement that is executed by the user when debugging an application program
- (D) A condition the system tests for the validity of the database user

5.8 Entity set that does not have enough _____ to form a _____ is a weak entity set.

- (A) attribute, primary key
- (B) records, foreign key
- (C) records, primary key
- (D) attribute, foreign key

QA