Data, Databases, and Data Science

kaggle.com

- Public datasets on almost any possible topic
- Notebooks with code for data analysis/science
- Courses on many topics, including SQL
- **Make an account and explore**
- **Use datasets in your project**

CSV Data Files

- 25K out of 50K (half of the kaggle datasets, by far the most)
- California cities dataset (https://www.kaggle.com/camnugent/california-housing-feature-engineering)

Structured Data – CSV Tables

- California cities
  - Columns: County, City, Incorporation_date, …
  - Tuples: (Merced, Merced, 1889, …)
  - Statistics per column: range, unique values, histogram, mode

Relational Databases

- Management of tables
  - Storage
  - Modification (insert, delete, update)
  - Query
  - Backup
  - Transactions (concurrent multi-user access)
- SQL
  - Programming language for tables

Semi-structured Data – XML, JSON

- JSON files (only 3K out of 50K)
  - ArXiv dataset (https://www.kaggle.com/Cornell-University/arxiv)
  - (key, value) pairs where the key is explicit
- NoSQL databases

Unstructured Data

- Text and anything else
- Webpages
- Data processing applications

Data Science

- Extract information/value from data
  - Statistics
  - Correlations
  - Trends
  - Models
  - Predictions (machine learning)

PANDAS

Pandas

- Python library to work with CSV tables
- Kaggle course: https://www.kaggle.com/learn/pandas

Workflow

- Create a panda object
- Read the file
  - read_csv
- Vector & Dictionary
  - Index by position
    - iloc, loc
  - Index by column name

- Operations
  - Select tuples (rows) based on attribute value
  - Create new column
  - Column statistics
  - GroupBy-Aggregate
  - Sort on columns
  - Missing values (isnull)
  - Column renaming
  - Join two pandas

Panda Programming in Python Notebooks

- Function calls for every operation
- Create new pandas from the input
- Imperative
  - Write code that tells what to do
- Interactive
  - Get the output of every operation (cell)
  - Visualization
  - Debugging

Relational Data Model

Types of Data

- Structured data
  - CSV tables
  - The largest category on kaggle.com
- Semi-structured data
  - JSON files
- Unstructured data
  - Text, web pages

Data Model

- Structure
- Values constraints
- Operations

Relational Data Model

- Data model for CSV tables
- Structure
  - TABLE or RELATION is the only element
- Value constraints
  - Unique or keys
  - NULLs
- Operations
  - Relational algebra or algebra for tables

TABLE Or Relation (1)

- Attributes or columns
  - Table header: name, latitude, longitude
  - Type or domain
    - Primitive: int, float, char[], string or varchar[]
    - Containers not allowed
- Schema
  - Cal_Cities (name, latitude, longitude)
- Tuples
  - (Merced, 37.302164, -120.482967)

TABLE Or Relation (2)

- Simple and general
  - (Any) Type of data can be represented as a table
- Abstract representation from implementation
  - Array (vector) of struct
  - Linked list of struct
  - Hash table of struct

Keys and NULLs

- Key
  - Attribute (or set of attributes) that have unique (different) values across all the tuples
  - There are no two different tuples which have the same value for the key attribute
  - Cal_Cities → name
- NULL
  - Missing value for an attribute in a tuple
  - Cal_Cities_Pop → pop_1980

Relational Algebra

- Set of operations on tables
  - A table is seen as a collection (or set) of tuples
  - Cannot index in the table
    - Cal_Cities[7] is not a valid operation
- Single table operations
  - Select column, select tuple (row), aggregate, grouping
- Multiple table operations
  - Product and Join, Union, Intersection, Difference

Schema Examples

- California_Cities
- Computers
- TPCH

SQL Data Definition Language (DDL)

CREATE TABLE (1)

CREATE TABLE **Product** (

maker char(32),

model integer,

type varchar(32)

)

- table/relation name
- attribute/column name and type
- Only creates the schema, without data

CREATE TABLE (2)

- No details about the implementation
  - What data structure?
    - Vector
    - Linked list
    - Hash table
  - What file format?
    - CSV
    - Binary
- High level of abstraction

SQLite CREATE TABLE (1)

- https://sqlite.org/lang_createtable.html
- Attribute/column data types
  - https://sqlite.org/datatype3.html
  - CHAR vs. VARCHAR
  - DECIMAL(tot_digits, decimal_digits)
  - DATE & DATETIME
    - https://sqlite.org/lang_datefunc.html

SQLite CREATE TABLE (2)

- DEFAULT
  - Default value of an attribute
- PRIMARY KEY
  - No duplicates are allowed for an attribute across all the tuples in the table
  - Only one per table
  - NULLs are allowed (because of a bug, not standard)
- UNIQUE
  - No duplicates are allowed for an attribute across all the tuples in the table
- NOT NULL
  - No empty values allowed

SQLite CREATE TABLE (3)

- ROWID
  - Unique integer associated with every row in a table
  - Not necessarily based on the row order
  - Created automatically by the system
- INTEGER PRIMARY KEY
  - Becomes the equivalent of ROWID

DROP TABLE

- CREATE TABLE
  - Register an empty table with the database
- DROP TABLE
  - Deletes the table from the database
  - **ALL DATA (TUPLES) are DELETED !!!**
- DROP TABLE **Product**

ALTER TABLE

- Modify the schema of a table
- ADD COLUMN
  - Adds a new column, without any value for existing tuples
  - ALTER TABLE **Cal_Cities_Pop** ADD COLUMN **pop_2020**
- DROP COLUMN
  - Removes a column, including all data across tuples
  - **NOT SUPPORTED IN SQLITE !!!**
- https://sqlite.org/lang_altertable.html

Examples

- California_Cities
- Computers
- TPCH

SQL Data Modification Operations

SQL CREATE TABLE

- Creates an empty table, with no data
  - Only the table header
  - No tuples (or rows)
- Similar to a struct declaration in C or class declaration in C++ / Java

SQL Modification Operations

- Add new tuples
  - INSERT INTO … VALUES
- Delete tuples
  - DELETE FROM … WHERE
- Update existing tuples with new values
  - UPDATE … SET … WHERE

INSERT

INSERT INTO **Product**

VALUES(**'A', 1001, 'pc'**)

INSERT INTO

PC(model, speed, ram, hd, price)

VALUES

(1001, 2.66, 1024, 250, 2114)

INSERT Examples

- 6.5.1 a)
  - INSERT INTO **Product(model, maker, type)**

    VALUES (**1100, ‘C’, ‘pc’**)
  - INSERT INTO **PC**

    VALUES (**1100, 3.2, 1024, 180, 2499**)

Bulk Loading

- Insert all the tuples from a CSV (text) file into a table
  - No INSERT statement for each tuple
- https://www.sqlite.org/cli.html
  - .mode “csv”
  - .separator “,”
  - .import csv_file table_name

DELETE

- DELETE FROM **Product**
- DELETE FROM **Printer** WHERE **color = false**
- 6.5.1 c)
  - DELETE FROM **PC** WHERE **hd < 100**

UPDATE

- UPDATE **Printer** SET **color = true**
- 6.5.1 e)
  - UPDATE **Product** SET **maker = 'A'** WHERE **maker = 'B'**
- 6.5.1 f)
  - UPDATE **PC** SET **ram = ram*2, hd = hd+60**

Examples

- California_Cities
- Computers
- TPCH

SQL Queries
Single Table

SQL Workflow

- CREATE TABLE
- INSERT TUPLES
  - Bulk load: .import
- **Queries**
  - **Data processing**
  - **Data analysis**
  - **Data science**

- PANDAS
  - Create panda object
  - Read CSV file
  - Call functions

SQL Queries

SELECT result_table_schema

FROM input_tables

[WHERE table_predicates AND join_conditions]

[GROUP BY grouping_attributes]

[ORDER BY sorting_attributes]

SQL Queries – Single Table

SELECT result_table_schema

FROM table

[WHERE table_predicates AND join_conditions]

Data from Table

- SQL
  - SELECT *

    FROM Cities_Population
  - * corresponds to the complete schema of the input table
- PANDAS
  - city_pop.head()

Column(s) from Table

- SQL
  - SELECT city
    FROM Cities_Population
  - SELECT city, county
    FROM Cities_Population
- PANDAS
  - city_pop["City"]

Rename Columns in Result

```
SELECT
    city,
    county,
    incorporated AS established,
    pop_2010 AS current_population
FROM
    Cities_Population
```

```
SELECT
    city,
    pop_2010 – pop_2000 AS population_increase
FROM
    Cities_Population
```

No Index Access in SQL

- SQL
  - Only value based access
- PANDAS
  - city_pop["City"][20]
  - city_pop.iloc[20]
  - city_pop.iloc[20][1]
  - city_pop.loc[:10, ['City','County']]

Conditions or Predicates

- SQL
  - SELECT

    *

    FROM

    Cities_Population

    WHERE

    county = ‘Merced’

- PANDAS
  - city_pop.loc[city_pop.County == 'Merced']

Complex Predicates

SELECT city, pop_2000, pop_2010

FROM

Cities_Population

WHERE

(county = ‘Merced’ OR county = ‘Stanislaus’) AND

pop_2010 > pop_2000

Predicates on Strings

- SELECT city

  FROM

  Cities_Population

  WHERE

  city LIKE ‘San %’
- SELECT city

  FROM

  Cities_Population

  WHERE

  city LIKE ‘San%’
- SELECT city

  FROM

  Cities_Population

  WHERE

  city LIKE ‘%San__ %’
- SELECT city

  FROM

  Cities_Population

  WHERE

  city LIKE ‘%San__%’

Check NULL Attributes

```sql
SELECT
    city,
    incorporated,
    pop_1980,
    pop_1990
FROM Cities_Population
WHERE
    county = 'Los Angeles' AND
    pop_1980 is null
```

```sql
SELECT city,
  case pop_1980 is null
    when true then pop_1990
    else pop_1990 - pop_1980
  end as change_1980_1990
FROM Cities_Population
WHERE county = 'Los Angeles'
```

ORDER BY Result

- SELECT city, pop_2010

  FROM Cities_Population

  ORDER BY

  pop_2010 [DESC]
- select county, city

  from Cities_Population

  order by county, city

SELECT

city,

pop_2010 - pop_2000 as change_2000_2010

FROM Cities_Population

ORDER BY

change_2000_2010 [desc]

Exercise 6.1.3

- Check the file in the lecture materials for all SQL statements
- Run all the queries on the sample database created and populated in the previous lectures
- f)

```
select model, hd
from pc
where speed = 3.2 and price < 2000
```

Examples

- California_Cities
- Computers
- TPCH

SQL Queries
Set Operations

Sets and Multi-sets (Bags)

- Sets
  - A = {1,2,3}
    - Only unique elements
  - select city

    from Cities_Population
  - Key attributes are sets

- Multi-sets or bags
  - A’ = {1,1,2,3,3}
    - There are duplicates
  - select county

    from Cities_Population
  - Attributes with duplicate values are bags

Operations on Sets and Multi-sets

- Sets
  - A = {1,2,3}, B = {1,3,5}
- Union
  - A U B = {1,2,3,5}
- Intersection
  - A ∩ B = {1,3}
- Difference
  - A – B = {2}
  - B – A = {5}

- Multi-sets or bags
  - A' = {1,1,2,3,3}
  - B' = {1,2,2,3,4}
- Union
  - A' U B' = {1,1,1,2,2,2,3,3,3,4}
- Intersection
  - A' ∩ B' = {1,2,3}
- Difference
  - A' – B' = {1,3}
  - B' – A' = {2,4}

SQL Multi-sets

- SQL works with multi-sets or bags
- SQL does not eliminate duplicates by default
- select county

  from Cities_Population

- Transform a multi-set to a set
- select **DISTINCT** county

  from Cities_Population

- Do not apply on keys because they are already sets!
- DISTINCT is an expensive operation that can increase query runtime quite significantly

SQL Set Operations

- Set
  - UNION
  - INTERSECT
  - EXCEPT
- A UNION B
  is equivalent to
  DISTINCT A
  UNION ALL
  DISTINCT B

- Multi-set
  - UNION ALL
  - Not supported
    - INTERSECT ALL
    - EXCEPT ALL

SQL Set Operations Requirement

- The schemas of the operands have to be exactly the same, including the name and the order of the attributes
- Use renaming with AS on the SELECT

UNION

- select maker

  from product

  where type = ‘pc’

  union

  select maker

  from product

  where type = ‘laptop’

- select maker

  from product

  where type = ‘pc’

  union all

  select maker

  from product

  where type = ‘laptop’

- select maker

  from product

  where type = 'pc' or type = 'laptop'

INTERSECT

- select maker

  from product

  where type = ‘pc’

  intersect

  select maker

  from product

  where type = ‘laptop’

- **This does not produce the correct result anymore!**
  - **select maker**

    from product

    where type = 'pc' and type = 'laptop'

EXCEPT

- select maker
  from product
  where type = ‘pc’
  except
  select maker
  from product
  where type = ‘laptop’

- select maker
  from product
  where type = ‘laptop’
  except
  select maker
  from product
  where type = ‘pc’
- **Incorrect!**
  - **select maker**
    from product
    where type = 'laptop' and type <> 'pc'

Multiple Attributes

```
select model, (speed+ram+hd)/price as score
from pc
union all
select model, (speed+ram+hd+screen)/price as score
from laptop
order by score desc
```

Examples

- Computers
- TPCH

SQL Queries
Full-Relation Operations

SQL Queries

SELECT **[DISTINCT] [SUM | COUNT | AVG]** result_table

FROM input_tables

[WHERE table_predicates]

[GROUP BY grouping_attributes

[HAVING agg_condition]]

[ORDER BY sorting_attributes]

[UNION [ALL]] [INTERSECT] [EXCEPT]

Duplicate Elimination DISTINCT

SELECT **[DISTINCT]** result_table

FROM input_tables

[WHERE table_predicates]

- Transform the result from a multi-set (bag) to a set
- It is an expensive operation!

DISTINCT

- SELECT county

  FROM Cities_Population
- SELECT DISTINCT county

  FROM Cities_Population

- select maker

  from product
- select distinct maker

  from product
- select maker, type

  from product
- select distinct maker, type

  from product

Aggregates Functions

SELECT **[SUM | COUNT | AVG | MIN | MAX](agg_attributes)**

FROM input_tables

[WHERE table_predicates]

- The output table has a single tuple (row) that contains the result of the aggregate function
- When a single aggregate is computed, the result is a single table cell (1 row and 1 column)
- PANDAS describe() function

Aggregate Queries Cities

- PANDAS describe()
- SELECT count(county)

  FROM Cities_Population
- SELECT count(DISTINCT county)

  FROM Cities_Population
- select count(*) as cnt,

  min(pop_2010) as min_pop,

  avg(pop_2010) as avg_pop,

  max(pop_2010) as max_pop

  from Cities_Population

- select max(pop_2010-pop_2000) as max_pop_increase,

  min(pop_2010-pop_2000) as max_pop_decrease,

  avg(pop_2010-pop_2000) as avg_pop_increase

  from Cities_Population

Aggregate Queries Computers

- select count(*)

  from product

  where maker = ‘A’
- select AVG(price)

  from PC
- select MIN(price), AVG(price), MAX(price)

  from laptop
- select min(speed), min(hd)

  from pc

  where price > 1000
- select count (distinct maker)

  from product

  where type = ‘pc’

GroupBy Aggregates

SELECT **grouping_atts, [SUM | COUNT | AVG | MIN | MAX](agg_attributes)**

FROM input_tables

[WHERE table_predicates]

[GROUP BY grouping_atts

[HAVING agg_condition]]

- Split input table into groups of tuples that have the same value for the grouping_atts
- Compute the aggregate functions for the tuples in every group
- Output a **single** tuple for every group: (grouping_atts, agg_functions)
- **HAVING** is a WHERE applied on the output
- WHERE is applied before the grouping

GroupBy Aggregates Cities

- select county,
   count(*) as no_city,
   min(pop_2010) as min_pop,
   avg(pop_2010) as avg_pop,
   max(pop_2010) as max_pop,
   sum(pop_2010) as total_pop
  from Cities_Population
  group by county

- select county,
   count(*) as no_city,
   min(pop_2010) as min_pop,
   avg(pop_2010) as avg_pop,
   max(pop_2010) as max_pop,
   sum(pop_2010) as total_pop
  from Cities_Population
  group by county
  having no_city >= 10
  order by no_city desc, total_pop desc

GroupBy Aggregates Computers

- select speed, avg(price) as avg_price

  from pc

  group by speed
- select speed, avg(price) as avg_price

  from pc

  where speed > 2

  group by speed

- select maker, count (distinct model)

  from product

  group by maker
- select maker, count (distinct model)

  from product

  where type = 'pc'

  group by maker
- select maker, count (distinct model) as models

  from product

  where type = 'pc'

  group by maker

  having models >= 3

Examples

- Cities
- Computers
- TPCH

SQL Queries
Joins over Two or More Tables

SQL Queries

SELECT [DISTINCT] [SUM | COUNT | AVG] result_table

FROM **$table_1$, $table_2$**

[WHERE table_predicates AND **join_conditions**]

[GROUP BY grouping_attributes

[HAVING agg_condition]]

[ORDER BY sorting_attributes]

[UNION [ALL]] [INTERSECT] [EXCEPT]

Cartesian Product

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- R x S(A,B) = {

  (1,1),(1,3),(1,4),

  (1,1),(1,3),(1,4),

  (2,1),(2,3),(2,4),

  (3,1),(3,3),(3,4)}
- The result consists of pairs of one element from R and one from S
- Every element from R is paired with every element from S
- The number of elements in R x S is |R|*|S|, i.e., the size of R multiplied by the size of S

- select *

  from R, S
- The schema of the result is the **union** of the R schema and the S schema

Cartesian Product Generalization

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- T{C} = {2,4}
- R x S(A,B) = {
  (1,1),(1,3),(1,4),
  (1,1),(1,3),(1,4),
  (2,1),(2,3),(2,4),
  (3,1),(3,3),(3,4)}
- select * from R, S

- R x S x T(A,B,C) = {
  (1,1,2),(1,3,2),(1,4,2),
  (1,1,2),(1,3,2),(1,4,2),
  (2,1,2),(2,3,2),(2,4,2),
  (3,1,2),(3,3,2),(3,4,2),
  (1,1,4),(1,3,4),(1,4,4),
  (1,1,4),(1,3,4),(1,4,4),
  (2,1,4),(2,3,4),(2,4,4),
  (3,1,4),(3,3,4),(3,4,4)}
- select * from R, S, T

Two-Table Join

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- R $\bowtie_{A=B}$ S = {
  (1,1),~~(1,3)~~,~~(1,4)~~,
  (1,1),~~(1,3)~~,~~(1,4)~~,
  ~~(2,1)~~,~~(2,3)~~,~~(2,4)~~,
  ~~(3,1)~~,**(3,3)**,~~(3,4)~~} = {(1,1),(1,1),(3,3)}
- Join condition between attributes from the two tables
- Only those tuples from the Cartesian product that satisfy the join condition are included in the result

- select * from R, S where **A = B**
- Condition does not have to be equality
- select * from R, S where **A > B**
  - **{(2,1), (3,1)}**

Multiple-Table Join

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- T{C} = {2,4}
- select * from R, S, T

  where **A=B and B>C**
- If there is no condition for a table, Cartesian product is performed for that table

- $R \bowtie_{A=B} S \bowtie_{B>C} T(A,B,C) = \{$

~~(1,1,2),(1,3,2),(1,4,2),~~

~~(1,1,2),(1,3,2),(1,4,2),~~

~~(2,1,2),(2,3,2),(2,4,2),~~

~~(3,1,2)~~,(3,3,2),~~(3,4,2)~~,

~~(1,1,4),(1,3,4),(1,4,4),~~

~~(1,1,4),(1,3,4),(1,4,4),~~

~~(2,1,4),(2,3,4),(2,4,4),~~

~~(3,1,4),(3,3,4),(3,4,4)~~} = {(3,3,2)}

Duplicate Attribute Names

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- select * from Product, PC
  - schema: (maker, **Product.model**, type, **PC.model**, speed, ram, hd, price)
  - select **Product.model**, maker, price from Product, PC
  - select **P.model**, maker, PC.price from **Product P**, PC

Join Query Examples

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- select * from Product P, PC

  where **P.model = PC.model**
- select P1.maker, PC.model AS pc_model, L.model AS laptop_model

  from **Product P1, Product P2**, PC, Laptop L

  where **P1.maker = P2.maker and P1.model = PC.model and P2.model = L.model and PC.price > L.price**
  - Find the (PCs, laptop) pairs produced by the same maker for which the PC price is larger than the laptop price
  - Multiple instances of a table can appear in a query. They have to be renamed as the attributes are renamed.

Abstract Evaluation Model

- select P1.maker, PC.model AS pc_model, L.model AS laptop_model

  from Product P1, Product P2, PC, Laptop L

  where P1.maker = P2.maker and P1.model = PC.model and P2.model = L.model and PC.price > L.price
- **For** each tuple P1 in table Product

  For each tuple P2 in table Product

  For each tuple PC in table PC

  For each tuple L in table Laptop

  if P1.maker = P2.maker and P1.model = PC.model and P2.model = L.model and PC.price > L.price

  then add(P1.maker, PC.model, L.model) to the result

Abstract Evaluation Model for General Queries

SELECT [DISTINCT] [SUM | COUNT | AVG] result_table

FROM $table_1$, $table_2$, …

[WHERE table_predicates AND join_conditions]

[GROUP BY grouping_attributes

[HAVING agg_condition]]

[ORDER BY sorting_attributes]

[UNION [ALL]] [INTERSECT] [EXCEPT]

- **The evaluation model for joins is first applied to the entire WHERE clause**
- **Everything else is evaluated on the result of the join evaluation**

Examples

- Computers
- TPCH

SQL Queries
Join Expressions

Cross Join

- select * from Product, PC
- select * from Product **cross join** PC
- The two statements are identical
- **cross join** is Cartesian product
- **cross join** is only *syntactic sugaring*

Join and Inner Join

- select * from Product P, PC where P.model=PC.model
- select * from Product P **join** PC **on** P.model=PC.model
- select * from Product P **inner join** PC **on** P.model=PC.model
- The three statements are identical
- **join** and **inner join** are only *syntactic sugaring*
- **Cross join, join, and inner join do not provide any additional functionality beyond what can be expressed in WHERE**

Natural Join

- select * from Product P, PC where P.model=PC.model
- select * from Product P **join** PC **on** P.model=PC.model
- select * from Product P **natural join** PC
- The three statements are almost identical
- **natural join** implies equality predicates between the attributes with the same name across the two tables
- select * from Product **natural join** Printer
- select * from Product P, Printer Pr where P.model = Pr.model and **P.type = Pr.type**
  - This is probably not intended
- Only one copy of the join attribute is kept in result since they are equal
  - {P.model, PC.model} → {model}

Outer Joins

R(A,B)

0	1
2	3
0	1
2	4
3	4

S(B,C)

0	1
2	4
2	5
3	4
0	2
3	4

R ⋈ S [natural join] (A,B,C)

2	3	4
2	3	4

R ⋈ S [full outer join] (A,B,C)

2	3	4
2	3	4
0	1	-
0	1	-
2	4	-
3	4	-
-	0	1
-	2	4
-	2	5
-	0	2

Left (Right) Outer Joins

R(A,B)

0	1
2	3
0	1
2	4
3	4

S(B,C)

0	1
2	4
2	5
3	4
0	2
3	4

$R \bowtie S$ [full outer join] (A,B,C)

2	3	4
2	3	4
0	1	-
0	1	-
2	4	-
3	4	-
-	0	1
-	2	4
-	2	5
-	0	2

$R \bowtie_L S$ [left outer join] (A,B,C)

2	3	4
2	3	4
0	1	-
0	1	-
2	4	-
3	4	-

$R \bowtie_R S$ [right outer join] (A,B,C)

2	3	4
2	3	4
-	0	1
-	2	4
-	2	5
-	0	2

SQLite

- Only **left outer join** is supported

- select * from Product P **left outer join** PC on P.model = PC.model
  where P.type = 'pc'
- select * from Product P **left outer join** PC on P.model = PC.model

- select * from Product P **natural left outer join** PC
- select * from Product P **natural left outer join** PC
  where P.type = 'pc'

Examples

- Computers
- TPCH

SQL Subqueries

Subqueries

- SQL queries take as input one or more tables and produce a table as result
- Decompose a complex query into simpler parts and then assemble them back together
- **Replace a table with a query (SELECT statement) in another query**

Scalar Subqueries

- Queries that return a single value (scalar) can be used in the WHERE clause for conditions
- select *

  from PC

  where price **= (select max(price) from PC)**

IN and NOT IN

- Check if a value is member in a set
- select maker

  from Product

  where type = ‘pc’ and

  maker **IN (select maker**

  from Product

  where type = ‘laptop’)

EXISTS and NOT EXISTS

- Check if a query returns tuples or not (empty set)
- select *

  from PC

  where **not exists**

  (select *

  from PC PC1

  where PC1.price > PC.price

)

```
select *
from PC
where not exists
    (select *
    from PC PC1
    where PC1.price > PC.price
    )
```

LIMIT Clause

- Limit the number of tuples in the result
- select maker, ram

  from Product P, PC

  where P.model = PC.model

  order by ram DESC

  LIMIT 1

- select maker, ram

  from Product P, PC

  where P.model = PC.model and

      not exists (select ram

          from PC PC1

          where PC1.ram > PC.ram)

Correlated Subqueries

- Use attributes from an outer query inside a subquery
- select *

  from **PC**

  where not exists

      (select *

      from PC PC1

      where PC1.price > **PC.price**

)

Subqueries in FROM

- Any query can be placed in FROM because it is a table
- select P.model, maker, **SQ.price**

  FROM Product P,

  (select model, price

  from PC

  where ram = (select max(ram) from PC)

) SQ

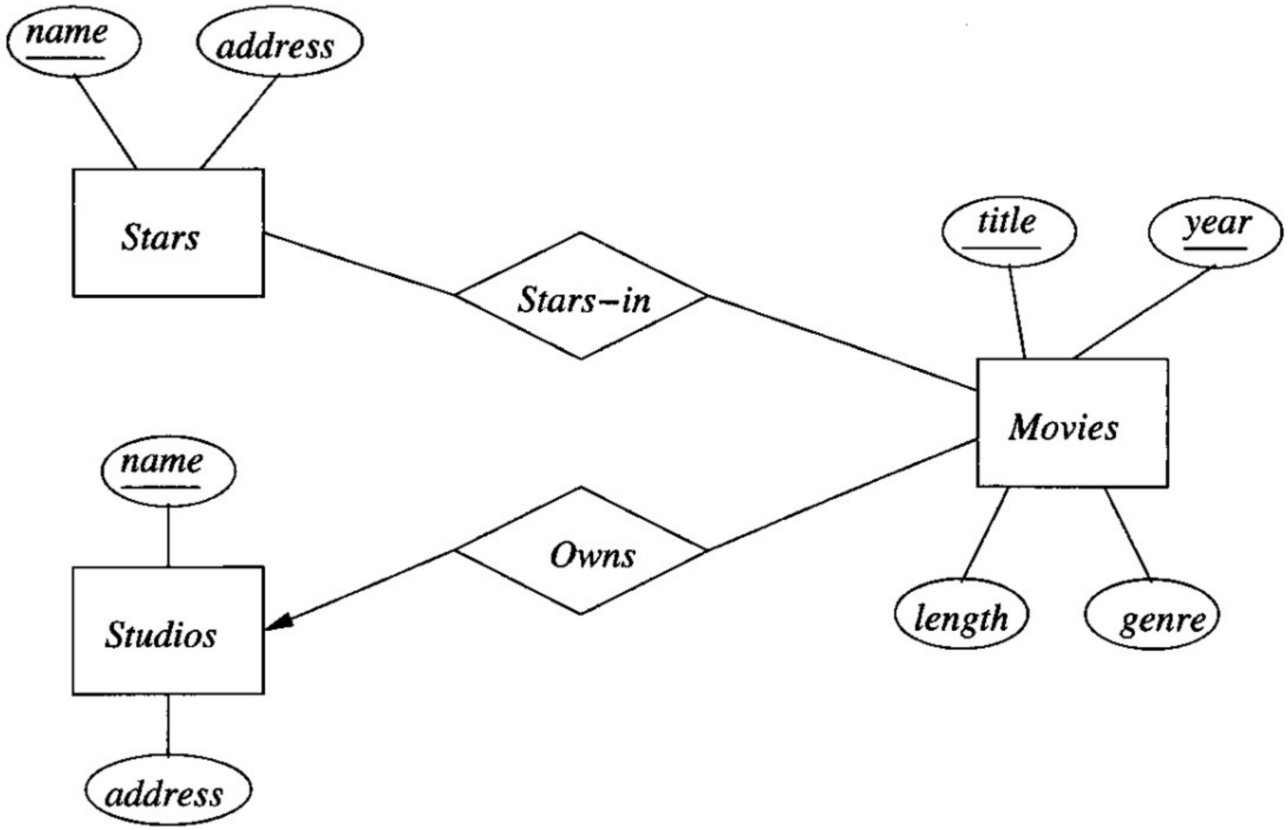
  where P.model = **SQ.model**

Examples

- Computers
- TPCH

E/R Diagrams
Mapping to Relations

E/R to Relations



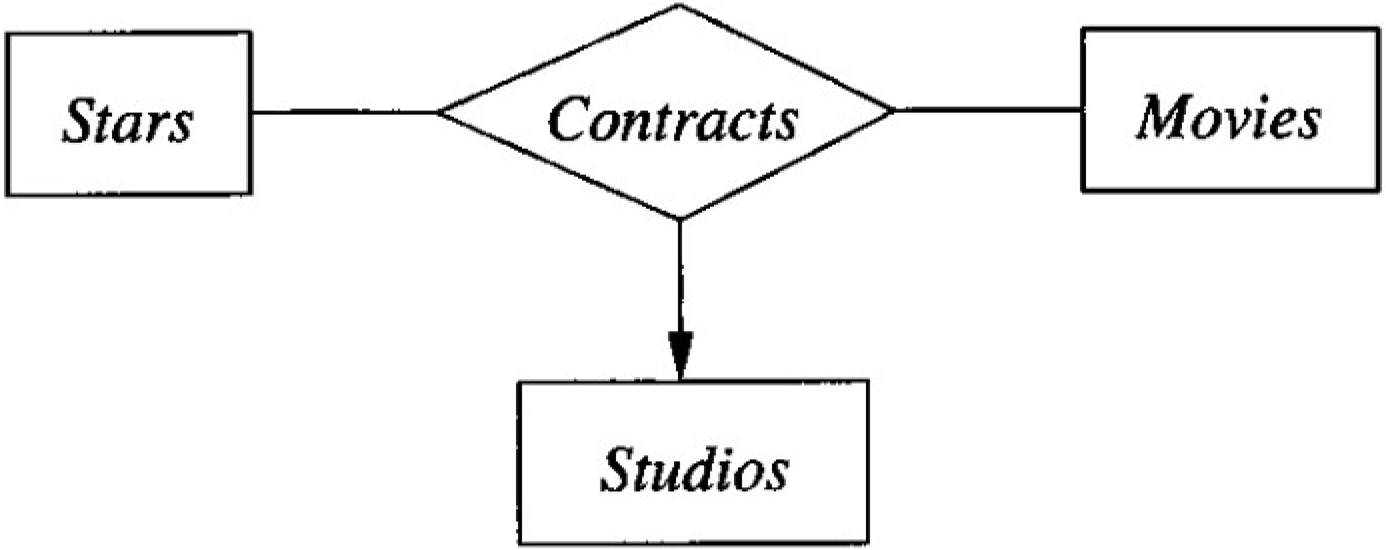
- Entities
  - Stars (**name**, address)
  - Movies (**title**, **year**, length, genre, **studioName**)
  - Studios (**name**, address)

- Many-to-many relationships
  - Stars-in (**starName**, **movieTitle**, **movieYear**)

One-to-one (-many) Relationships

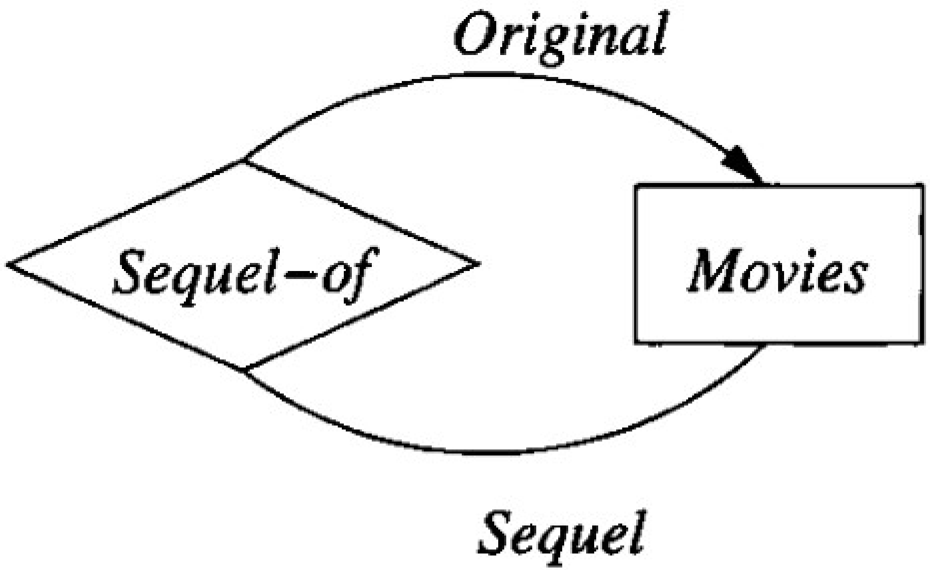
- Studios (**<u>name</u>**, address, **presidentName**)
- Presidents (**<u>name</u>**, **studioName**)

Multi (Three)-Way Relationships



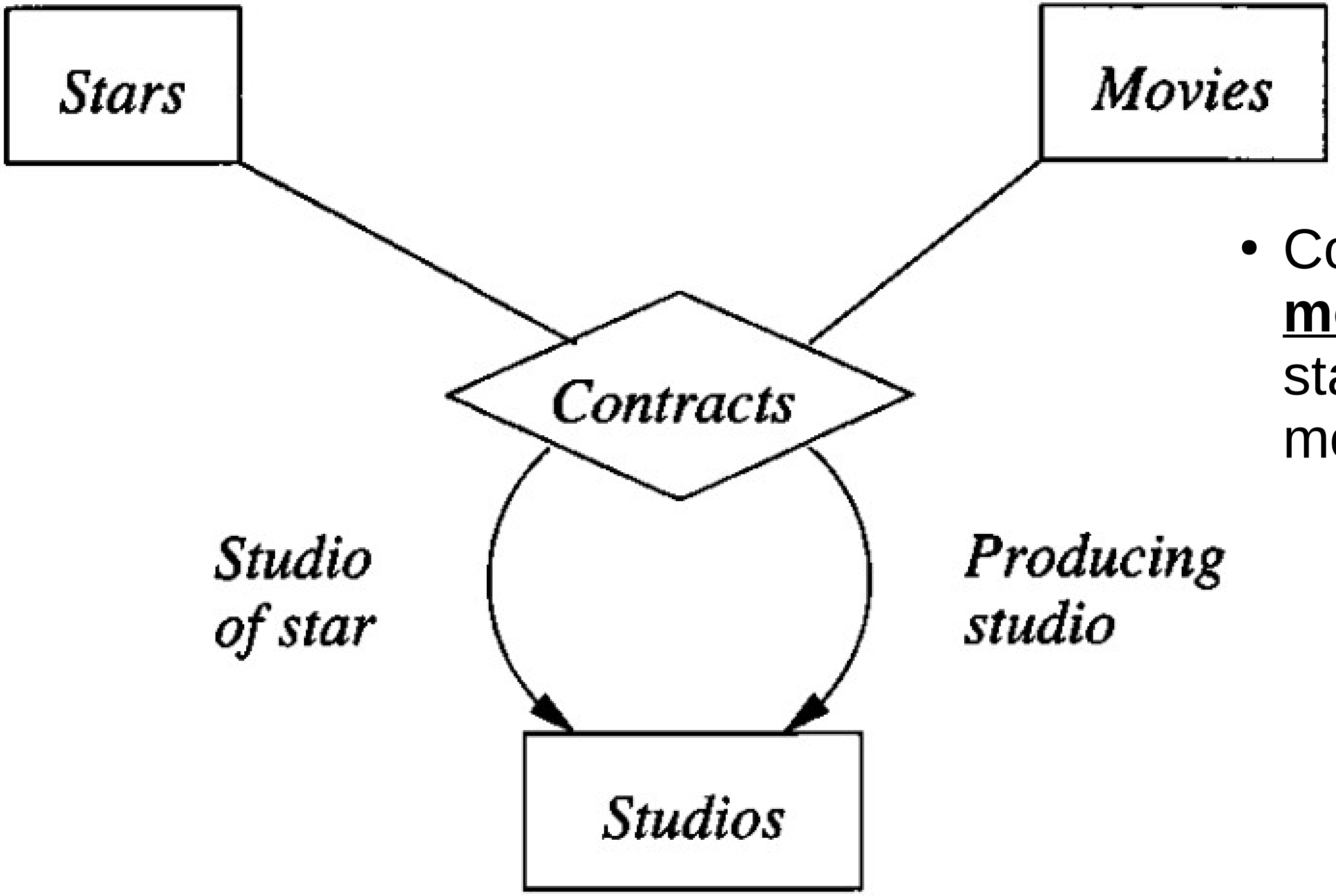
- Contracts (**<u>starName</u>**, **<u>movieTitle</u>**, **<u>movieYear</u>**, studioName)

Relationship with Roles



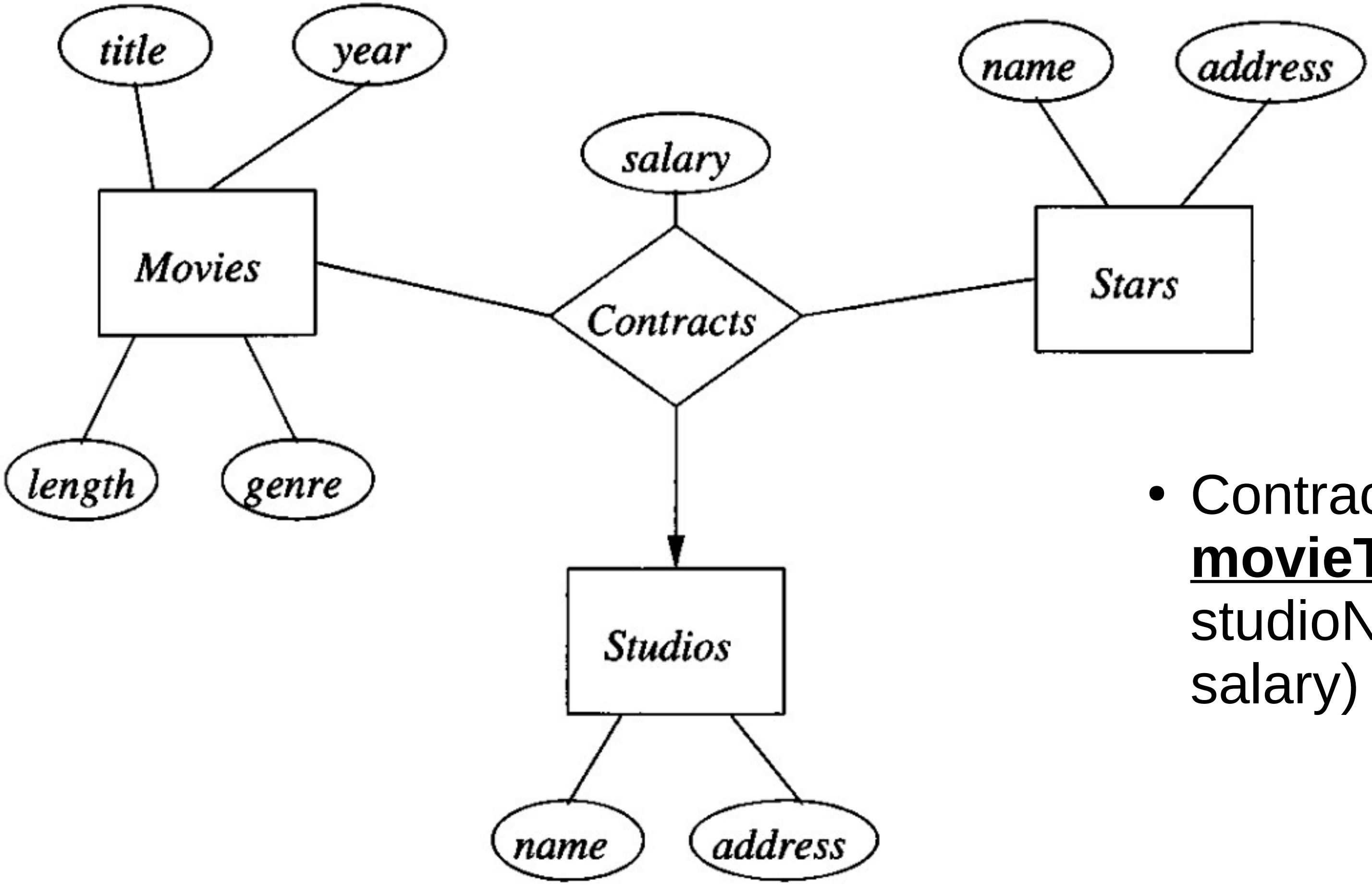
- Movies (**<u>title</u>**, **<u>year</u>**, length, genre, studioName, **originalTitle**, **originalYear**)

Multi (Four)-Way Relationships



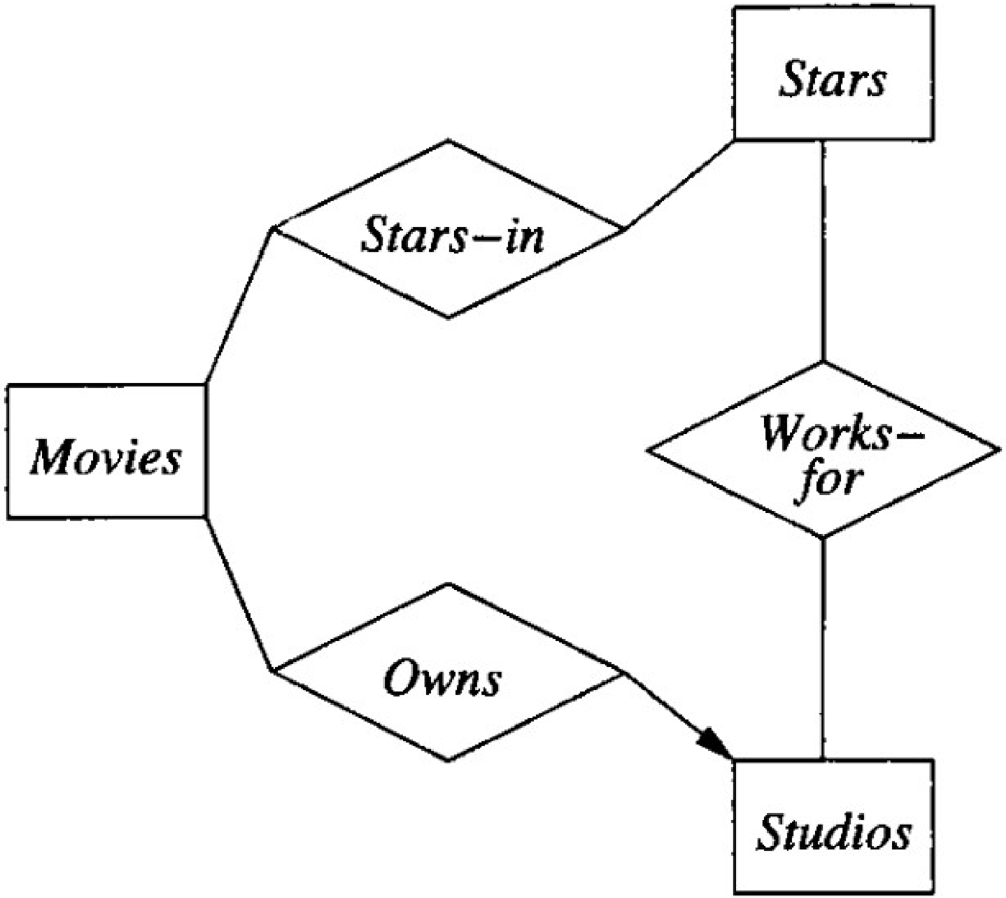
- Contracts (<u>**starName**</u>, <u>**movieTitle**</u>, <u>**movieYear**</u>, starStudioName, movieStudioName)

Relationships with Attributes



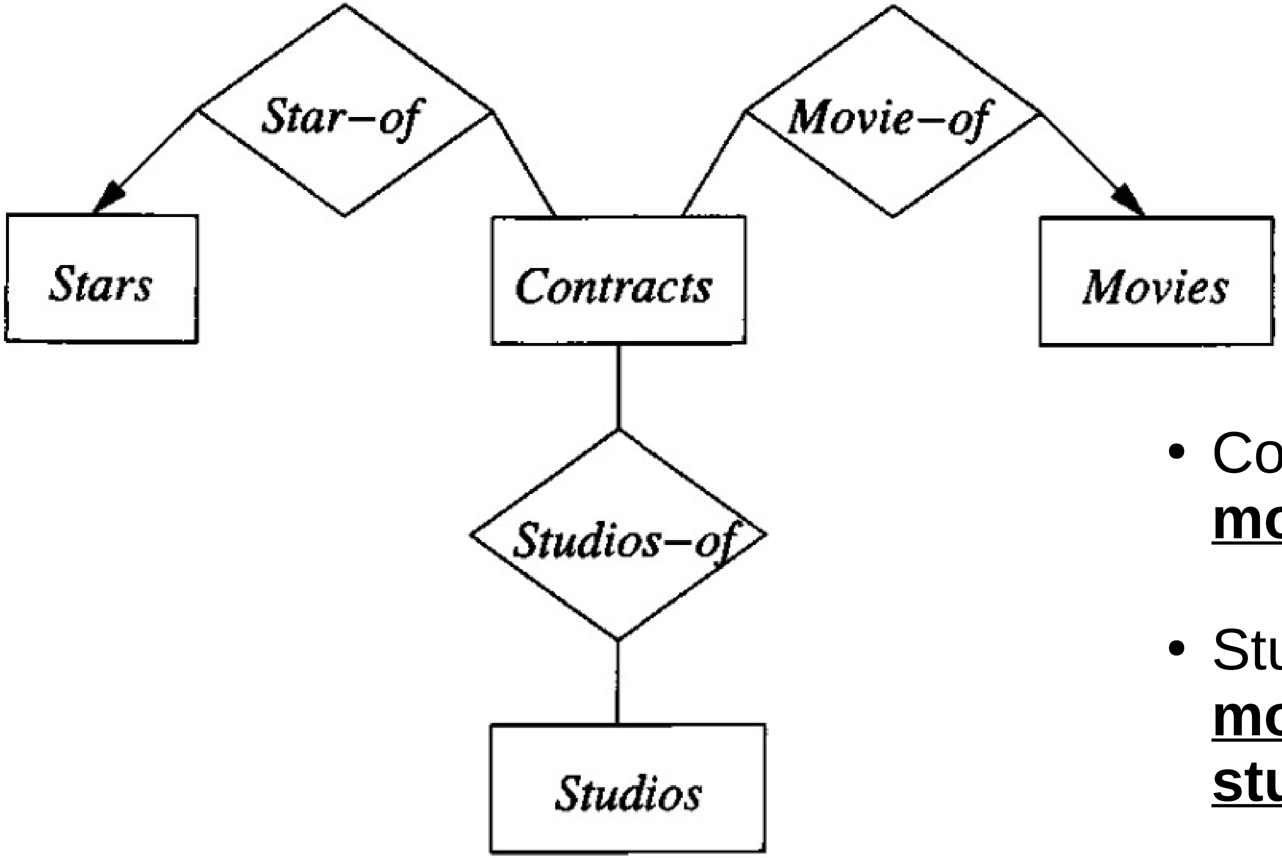
- Contracts (<u>**starName**</u>, <u>**movieTitle**</u>, <u>**movieYear**</u>, studioName, salary)

E/R to Relations



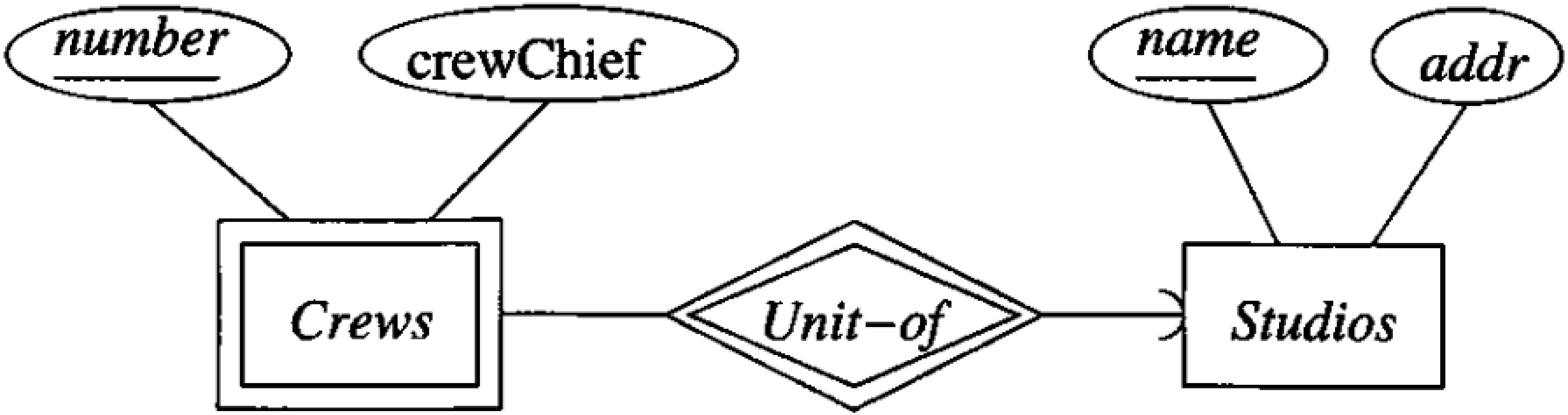
- Works-for (**<u>starName</u>**, **<u>studioName</u>**)

Multi-Way Relationships



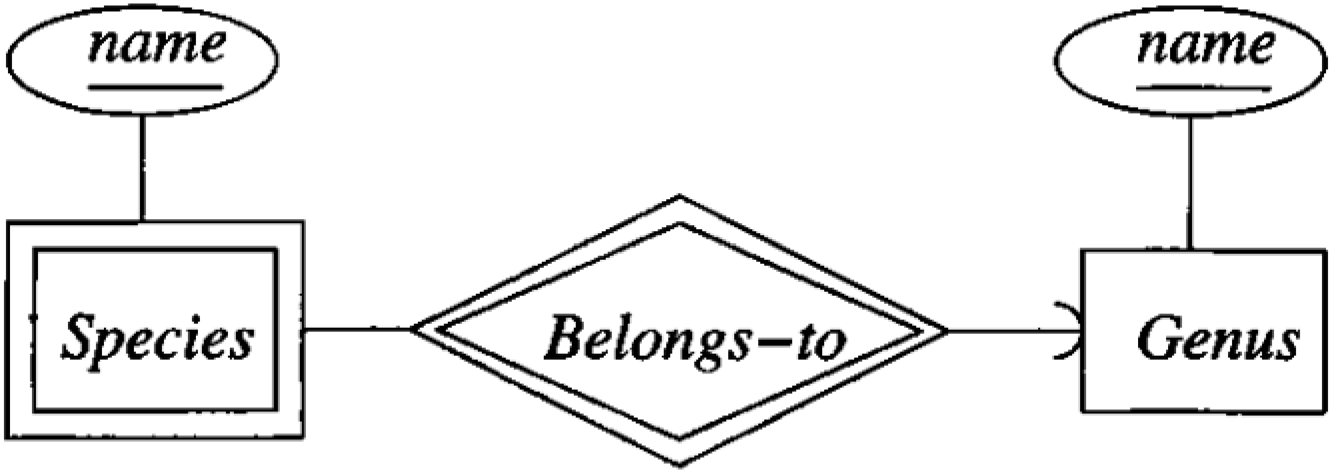
- Contracts (**<u>starName</u>**, **<u>movieTitle</u>**, **<u>movieYear</u>**)
- Studios-of (**<u>starName</u>**, **<u>movieTitle</u>**, **<u>movieYear</u>**, **<u>studioName</u>**)

Weak Entities (1)



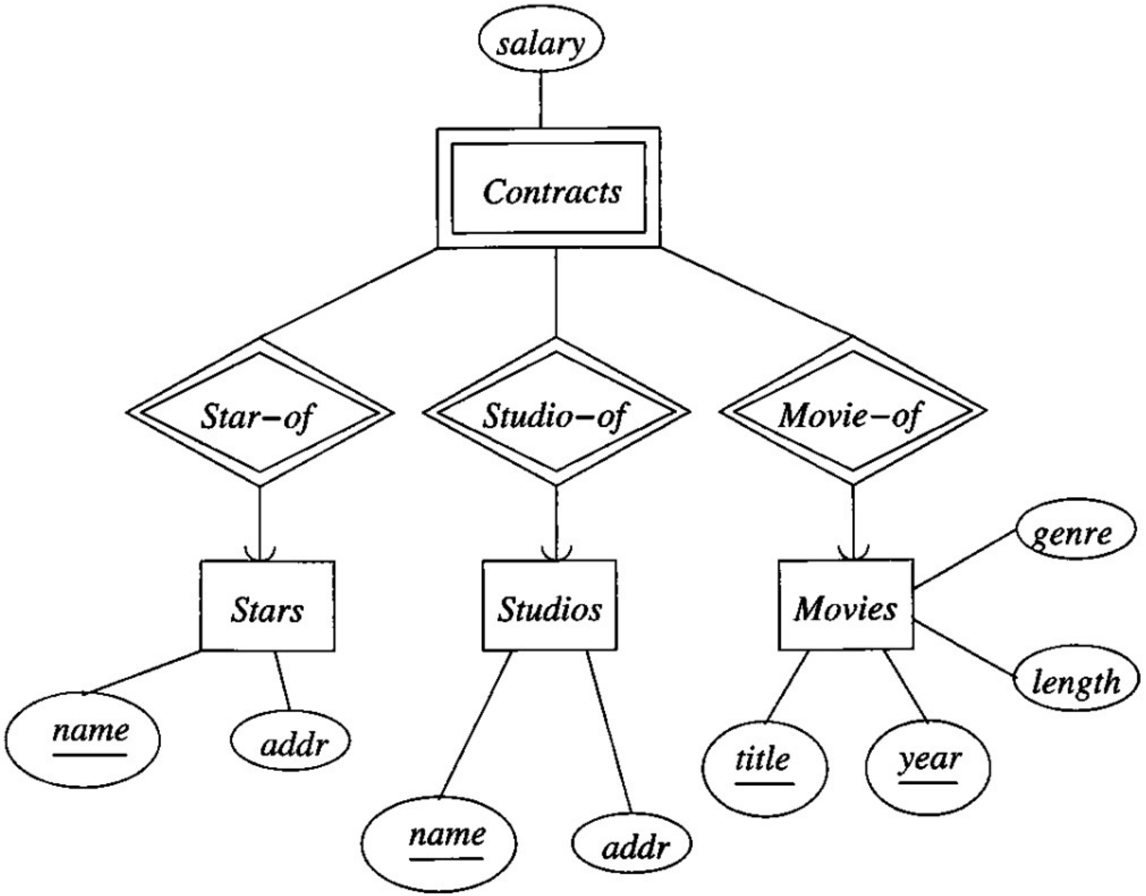
- Studios (**<u>name</u>**, addr)
- Crews (**<u>studioName</u>**, **<u>number</u>**, crewChief)

Weak Entities (2)



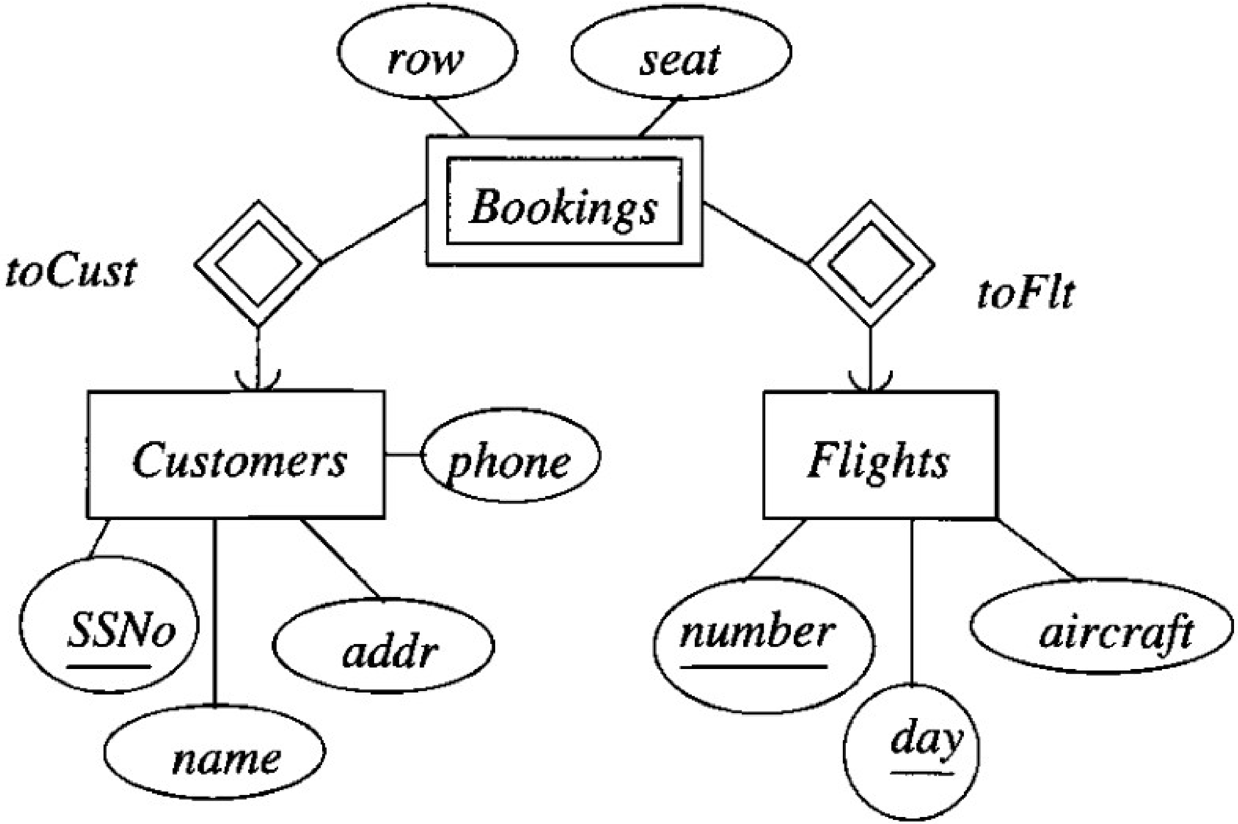
- Genus (**<u>name</u>**)
- Species (**<u>genusName</u>**, **<u>speciesName</u>**)

Weak Entities (3)



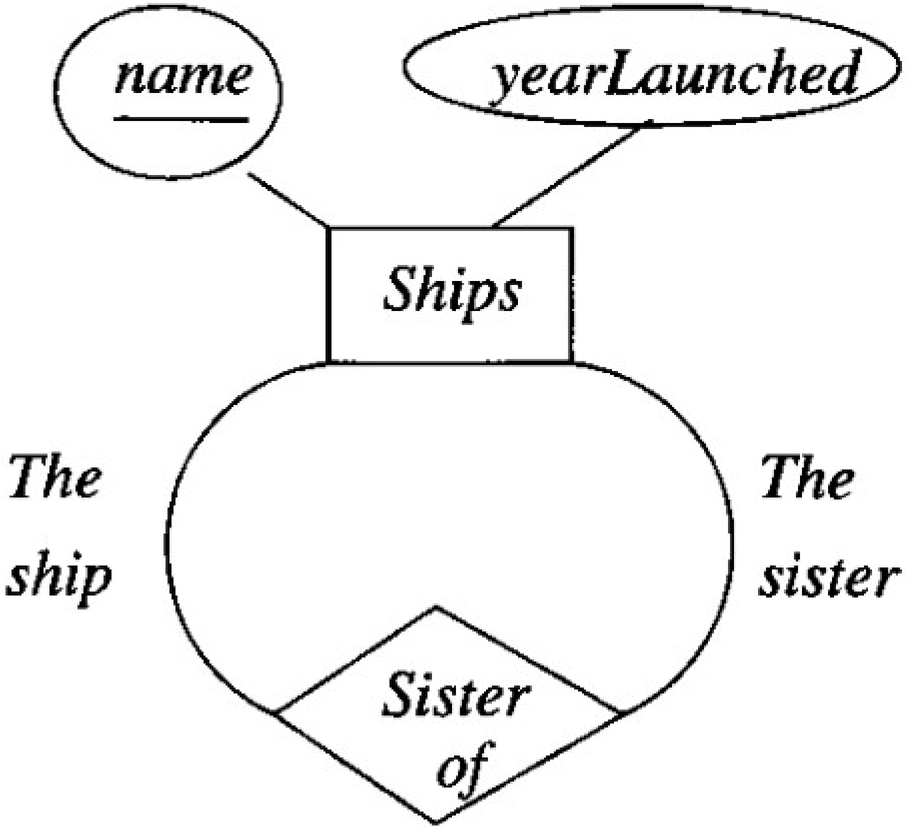
- Stars (**name**, addr)
- Studios (**name**, addr)
- Movies (**title**, **year**, genre, length)
- Contracts (**starName**, **studioName**, **movieTitle**, **movieYear**, salary)

Example (1)



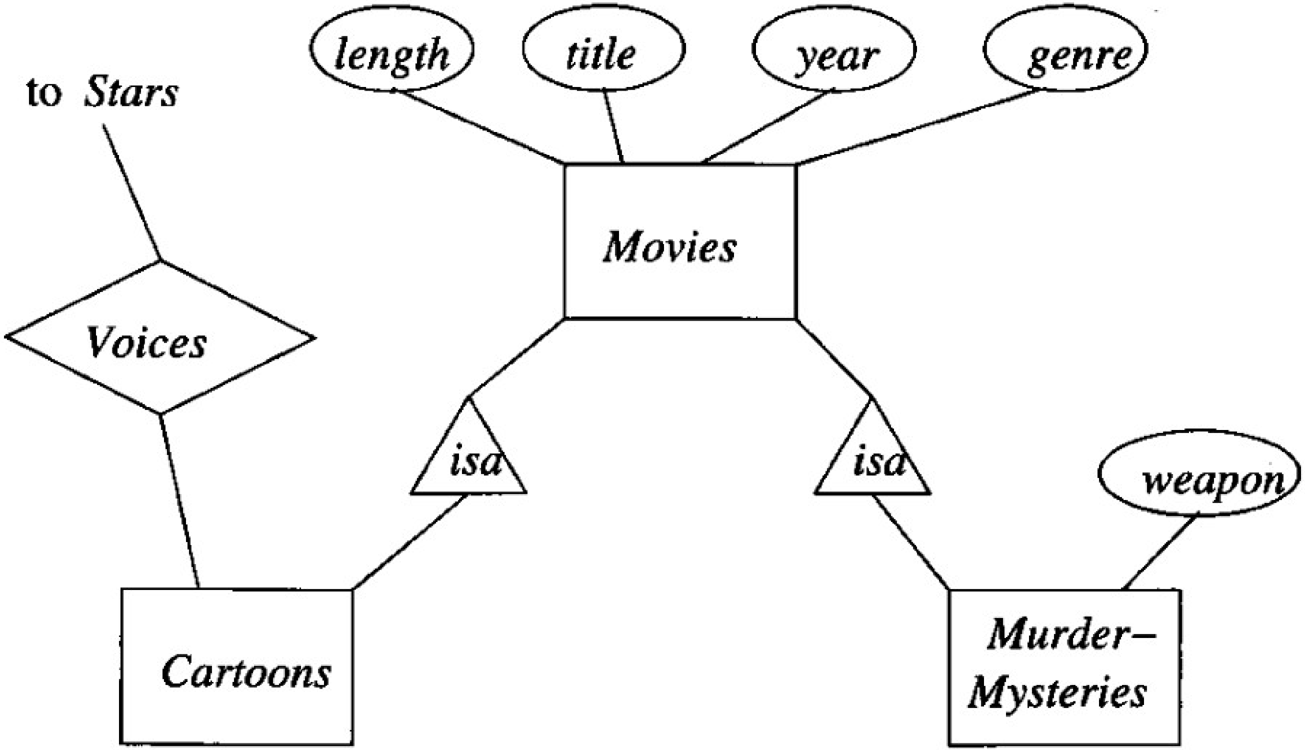
- Customers (**<u>SSNo</u>**, name, addr, phone)
- Flights (**<u>number</u>**, **<u>day</u>**, aircraft)
- Bookings (**<u>custSSNo</u>**, **<u>flightNo</u>**, **<u>flightDay</u>**, row, seat)

Example (2)



- Ships (**<u>name</u>**, yearLaunched)
- Sister-of (**<u>shipName</u>**, **<u>sisterShipName</u>**)

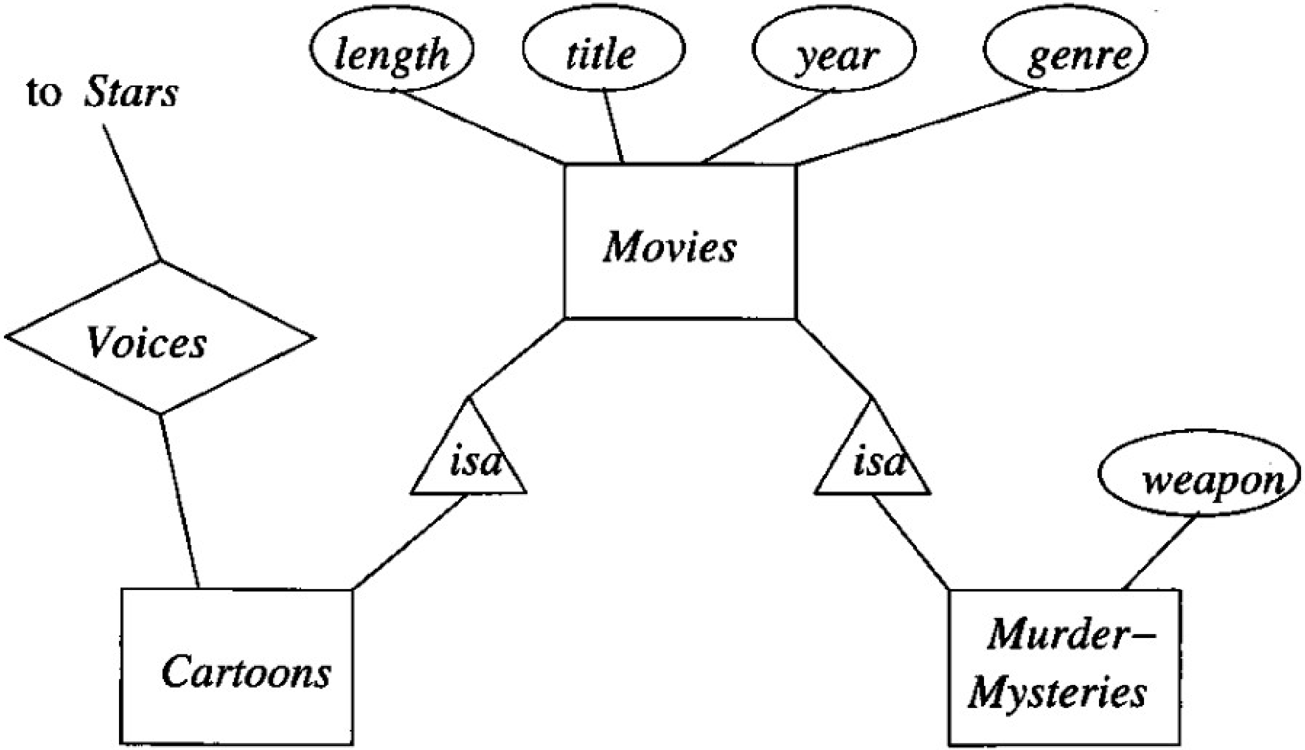
ISA Relationships: E/R Style



- Movies (**title**, **year**, genre, length)
- Cartoons (**title**, **year**)
- Murder-Mysteries (**title**, **year**, weapon)

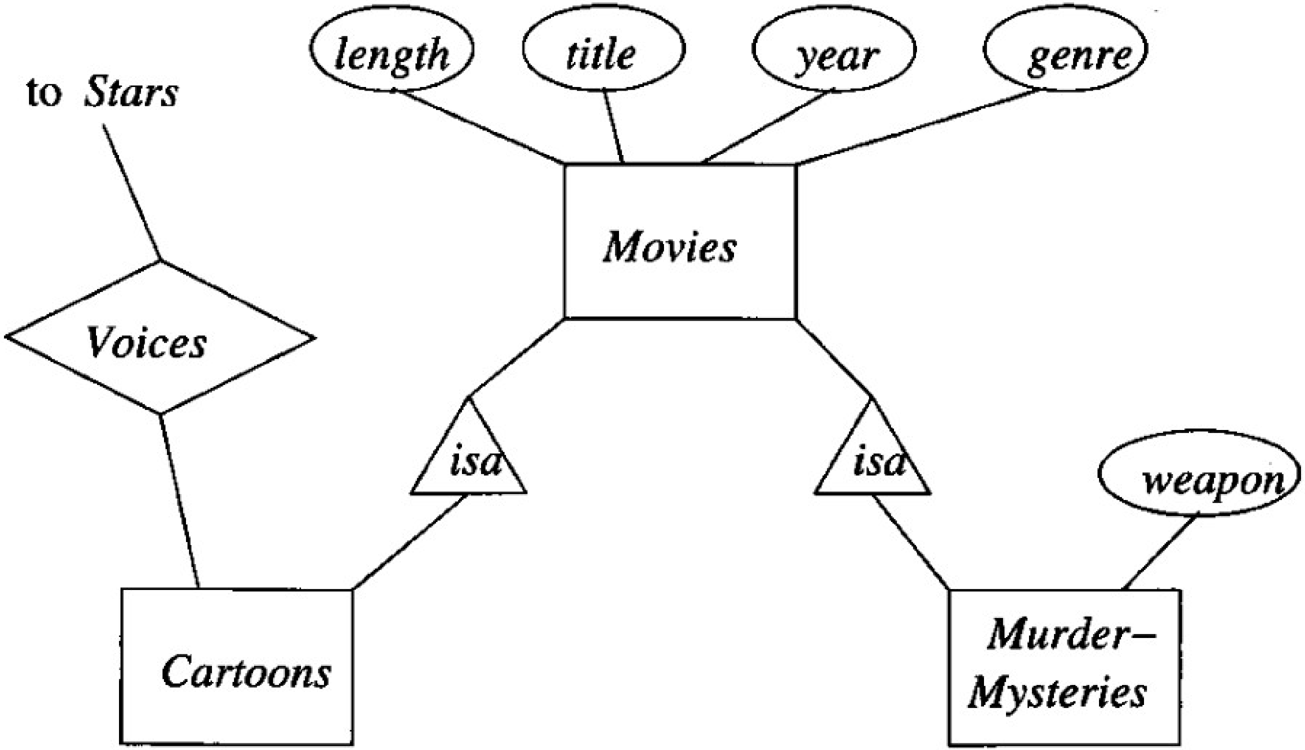
- Stars (**name**, address)
- Voices (**title**, **year**, **starName**)

ISA Relationships: Object-Oriented



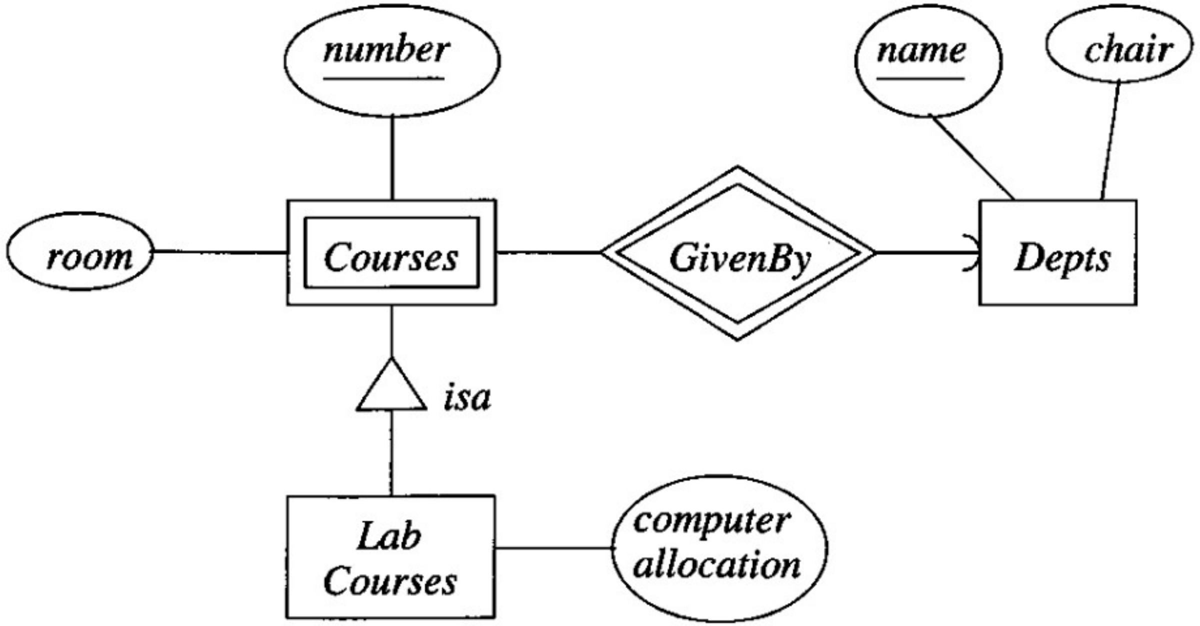
- Movies (<u>**title**</u>, <u>**year**</u>, genre, length)
- Cartoons (<u>**title**</u>, <u>**year**</u>, genre, length)
- Murder-Mysteries (<u>**title**</u>, <u>**year**</u>, genre, length, weapon)
- Cartoons-Murder-Mysteries (<u>**title**</u>, <u>**year**</u>, genre, length, weapon)

ISA Relationships: NULLs



- Movies (**<u>title</u>**, **<u>year</u>**, genre, length, weapon)

Example (3)



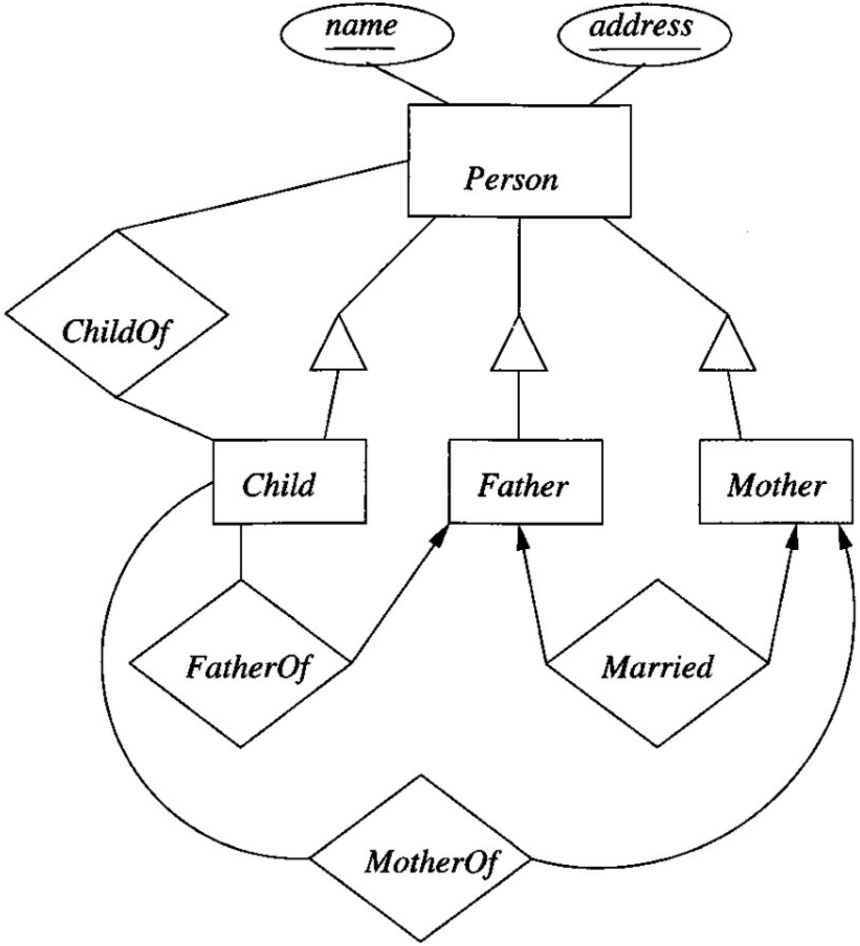
- Depts (**<u>name</u>**, chair)

- E/R style
  - Courses (**<u>deptName</u>**, **<u>number</u>**, room)
  - LabCourses (**<u>deptName</u>**, **<u>number</u>**, computerAllocation)

- Object-oriented
  - Courses (**<u>deptName</u>**, **<u>number</u>**, room)
  - LabCourses (**<u>deptName</u>**, **<u>number</u>**, room, computerAllocation)

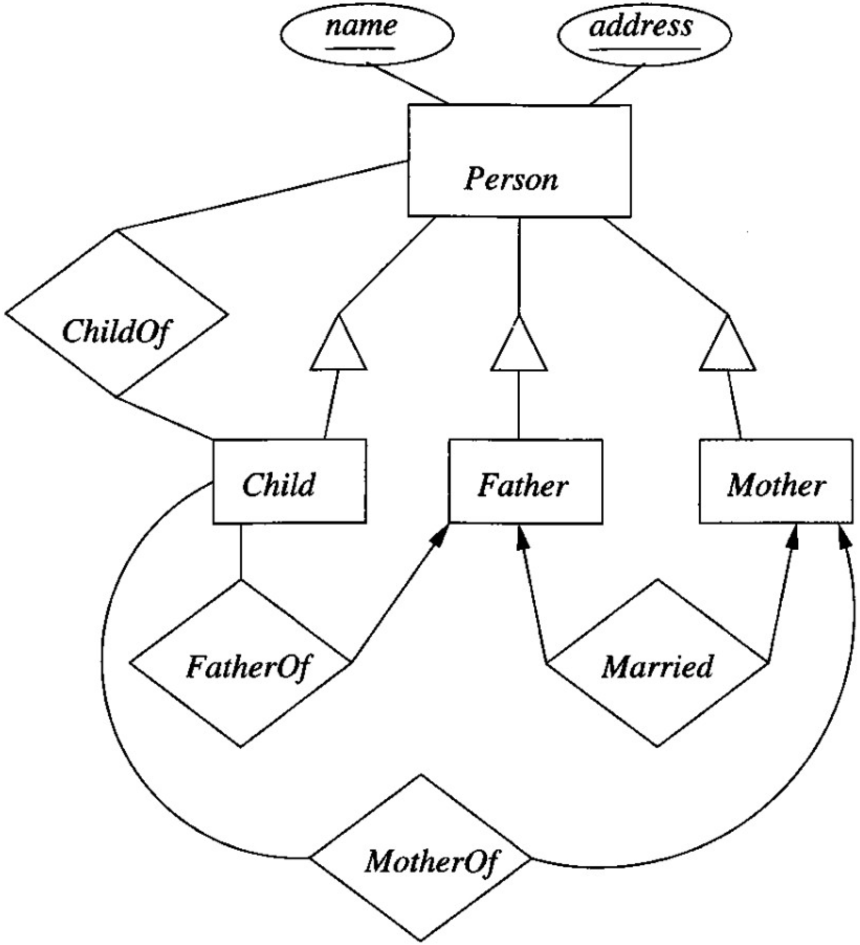
- NULLs
  - Courses (**<u>deptName</u>**, **<u>number</u>**, room, computerAllocation)

Example (4) E/R-style



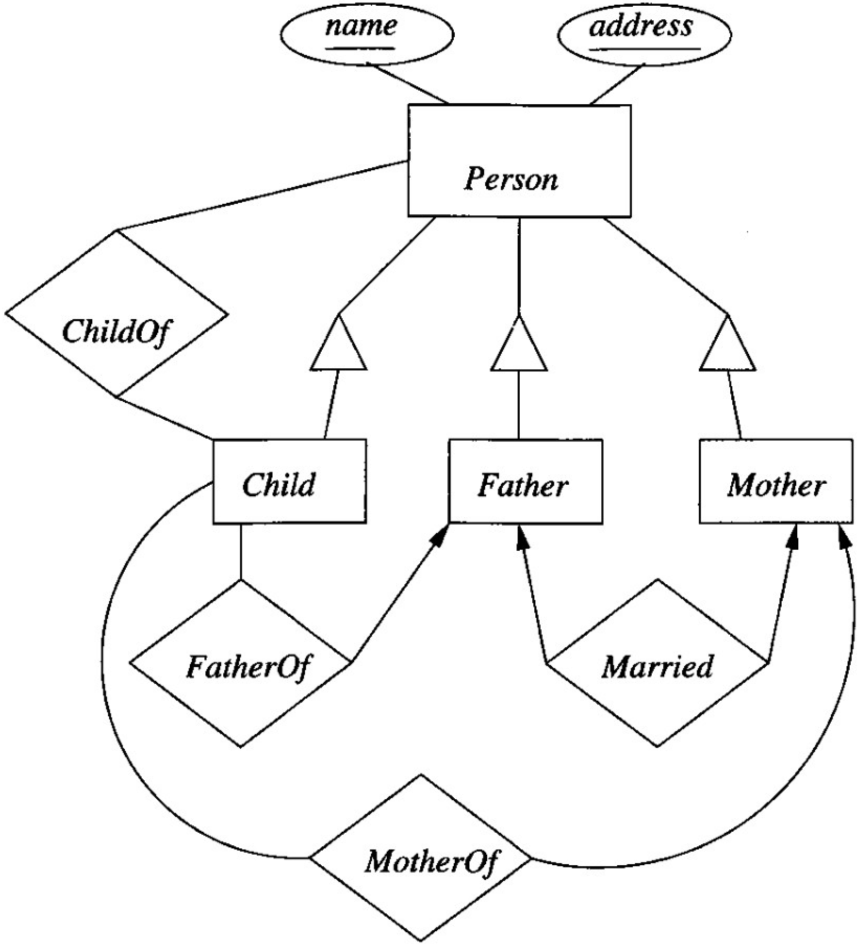
- Person (**name**, **address**)
- Child (**name**, **address**, fName, fAddr, mName, mAddr)
- Father (**name**, **address**, spouseName, spouseAddr)
- Mother (**name**, **address**)
- ChildOf (**pName**, **pAddr**, **cName**, **cAddr**)

Example (4) Object-Oriented



- Person (**<u>name</u>**, **<u>address</u>**)
- Child (**<u>name</u>**, **<u>address</u>**, fName, fAddr, mName, mAddr)
- Father (**<u>name</u>**, **<u>address</u>**, spouseName, spouseAddr)
- Mother (**<u>name</u>**, **<u>address</u>**)
- ChildFather (**<u>name</u>**, **<u>address</u>**, fName, fAddr, mName, mAddr, spouseName, spouseAddr)
- ChildMother (**<u>name</u>**, **<u>address</u>**, fName, fAddr, mName, mAddr)

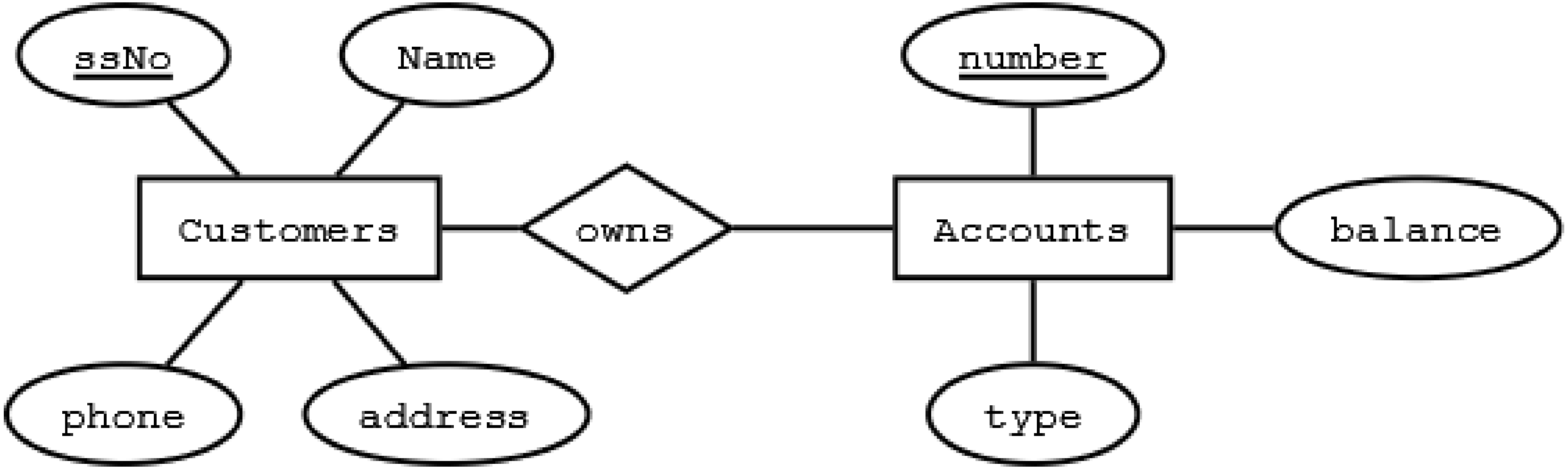
Example (4) NULLs



- Person (**<u>name</u>**, **<u>address</u>**, fName, fAddr, mName, mAddr, spouseFName, spouseFAddr, spouseMName, spouseMAddr)

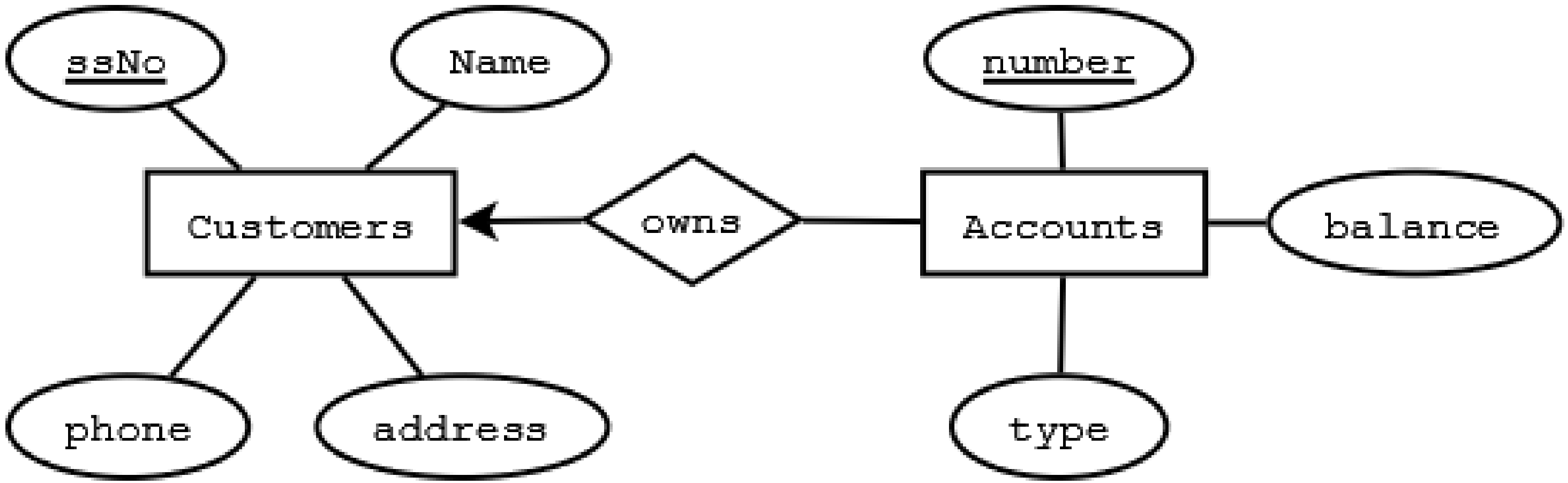
E/R Diagrams
Examples

Exercise 4.1.1



- Customers (**<u>ssNo</u>**, name, phone, address)
- Accounts (**<u>number</u>**, type, balance)
- Owns (**<u>ssNo</u>**, **<u>acctNo</u>**)

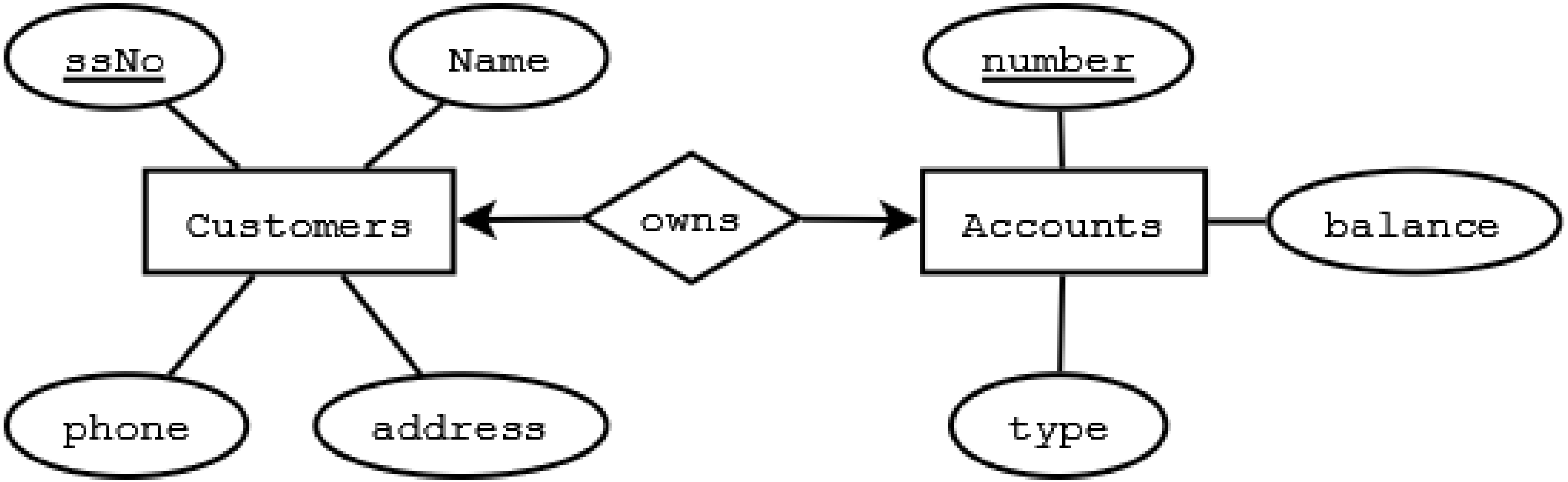
Exercise 4.1.2 a



- Accounts (**<u>number</u>**, type, balance, **ssNo**)
- Customers (**<u>ssNo</u>**, name, phone, address)

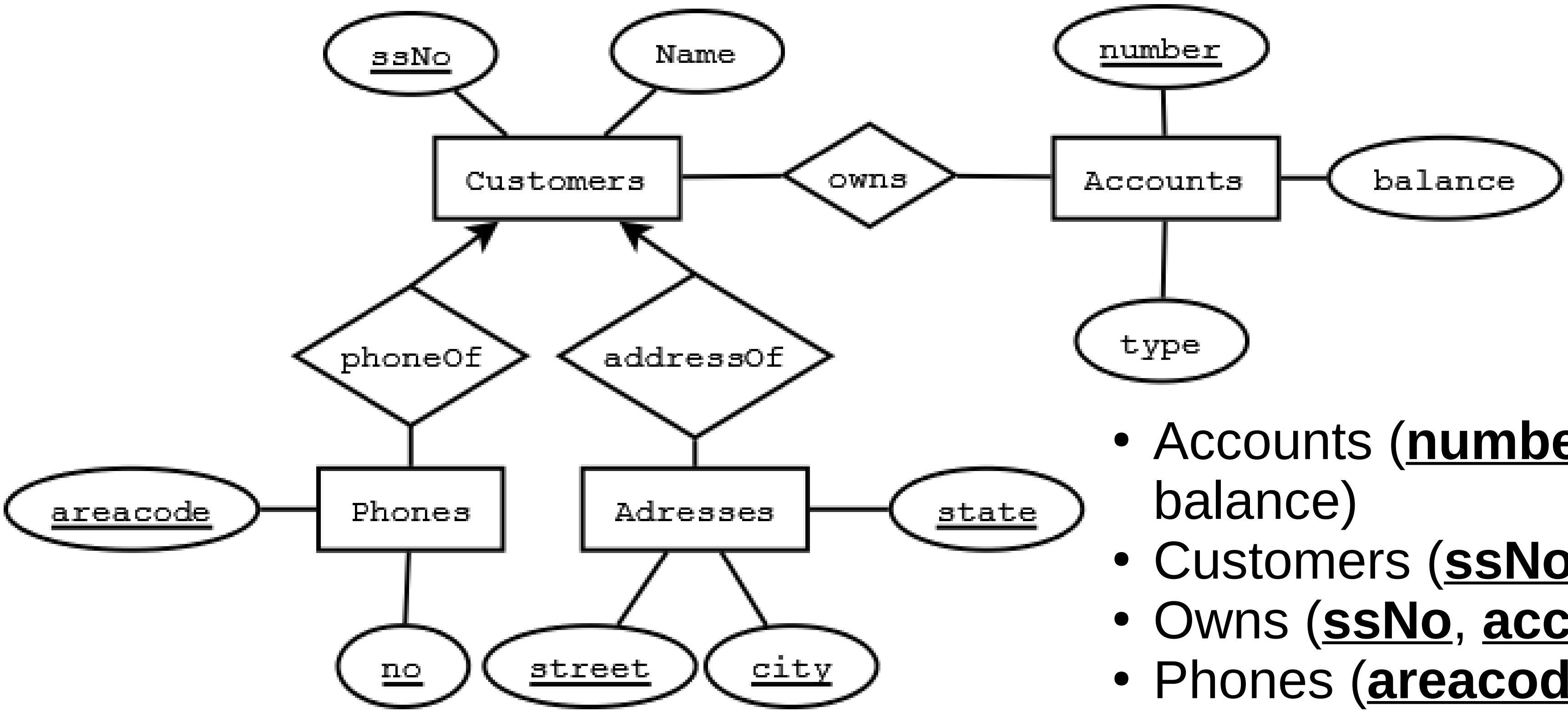
Exercise 4.1.2 b

- Accounts (**<u>number</u>**, type, balance, **ssNo**)



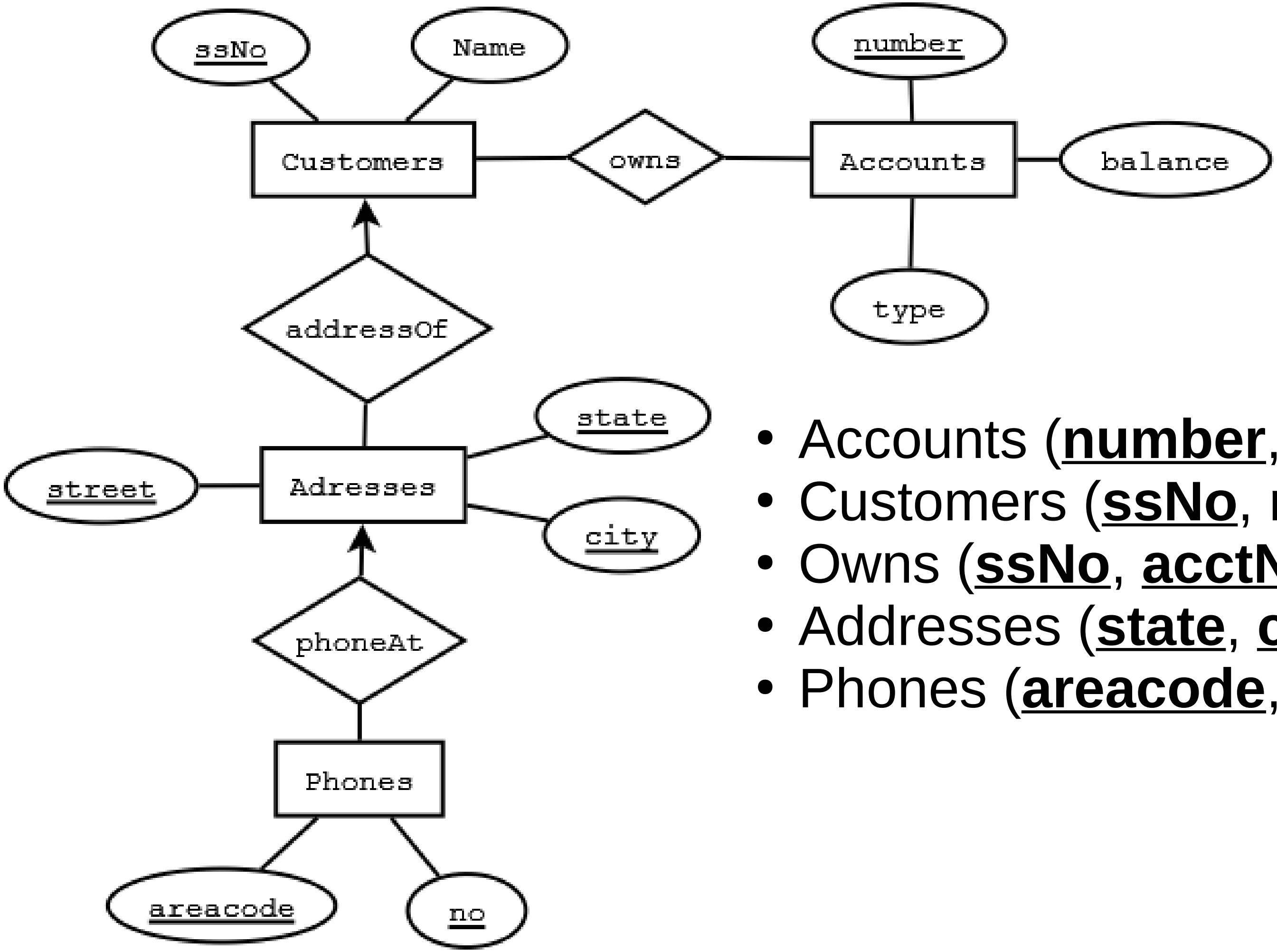
- Customers (**<u>ssNo</u>**, name, phone, address, **acctNumber**)

Exercise 4.1.2 c



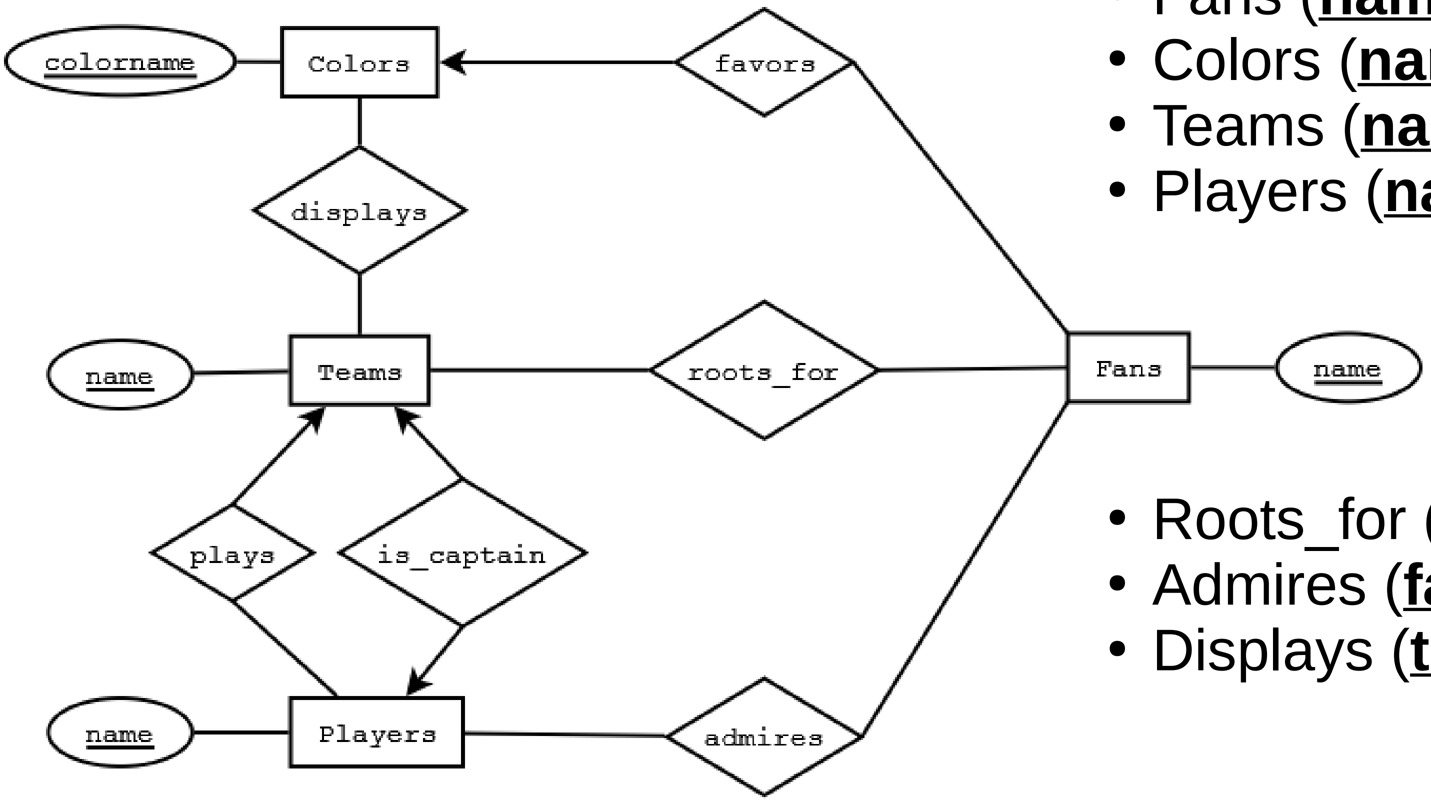
- Accounts (**number**, type, balance)
- Customers (**ssNo**, name)
- Owns (**ssNo**, **acctNo**)
- Phones (**areacode**, **no**, **ssNo**)
- Addresses (**state**, **city**, **street**, **ssNo**)

Exercise 4.1.2 d



- Accounts (**number**, type, balance)
- Customers (**ssNo**, name)
- Owns (**ssNo**, **acctNo**)
- Addresses (**state**, **city**, **street**, **ssNo**)
- Phones (**areacode**, **no**, **state, city, street**)

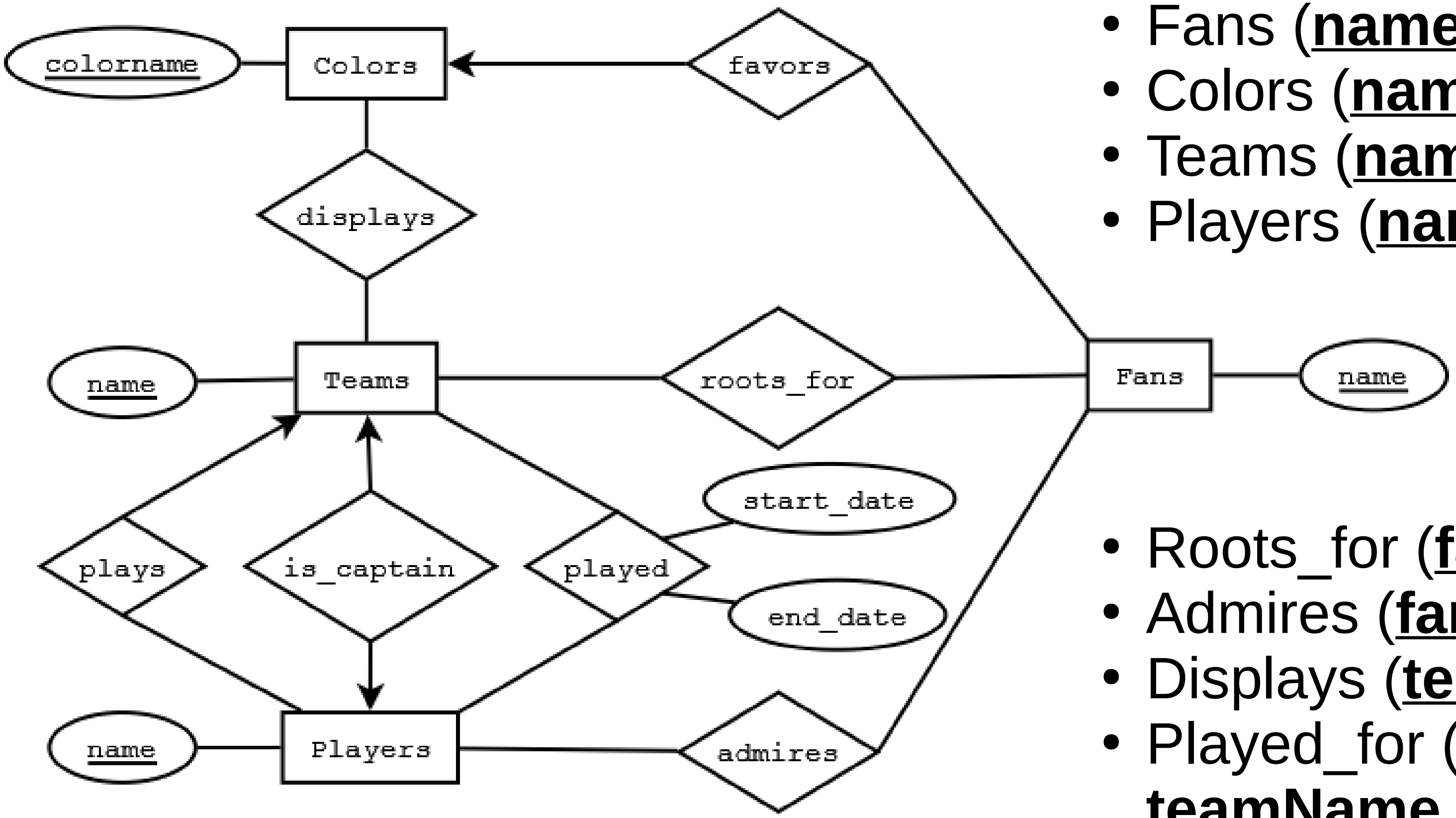
Exercise 4.1.3



- Fans (**<u>name</u>**, **favoriteColor**)
- Colors (**<u>name</u>**)
- Teams (**<u>name</u>**, **captainPlName**)
- Players (**<u>name</u>**, **teamName**)

- Roots_for (**<u>fanName</u>**, **<u>teamName</u>**)
- Admires (**<u>fanName</u>**, **<u>playerName</u>**)
- Displays (**<u>teamName</u>**, **<u>colorName</u>**)

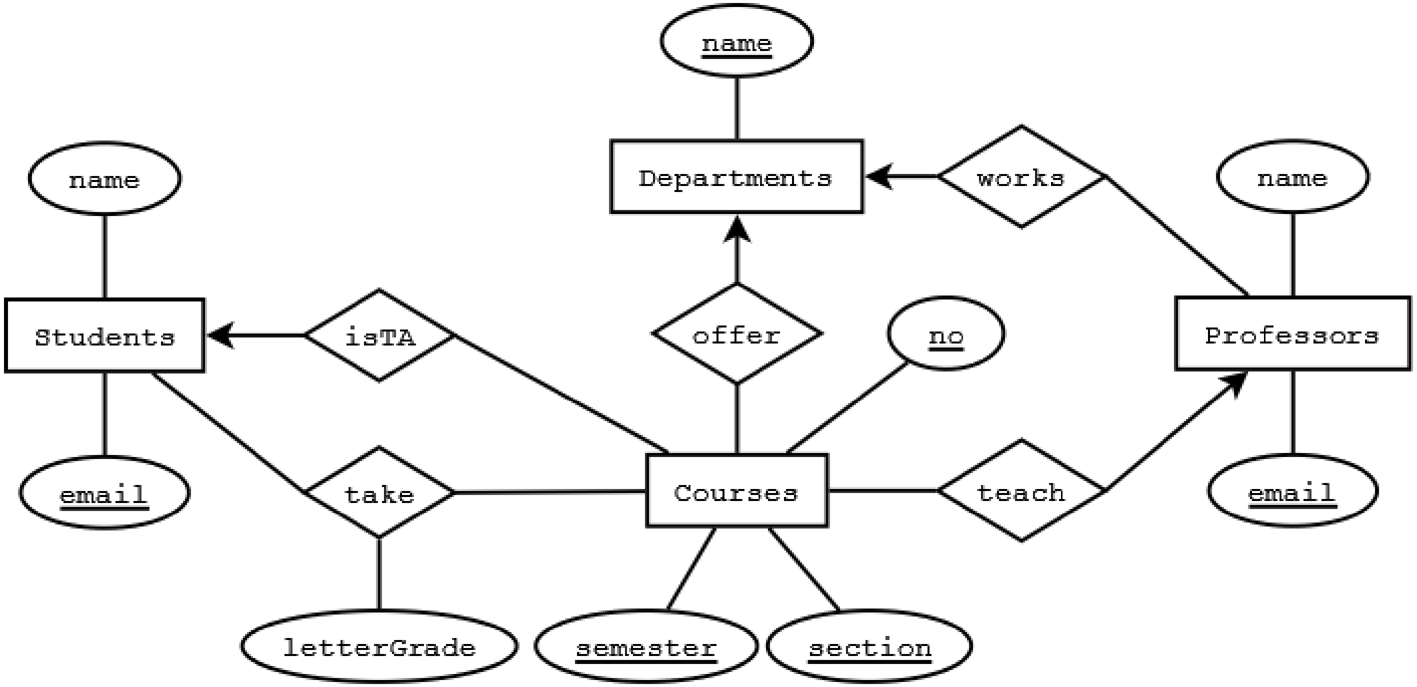
Exercise 4.1.5



- Fans (**name**, **favoriteColor**)
- Colors (**name**)
- Teams (**name**, **captainPlName**)
- Players (**name**, **teamName**)

- Roots_for (**fanName**, **teamName**)
- Admires (**fanName**, **playerName**)
- Displays (**teamName**, **colorName**)
- Played_for (**playerName**, **teamName**, start_date, end_date)

Exercise 4.1.9



- Students (**<u>email</u>**, name)
- Courses (**<u>no</u>**, **<u>semester</u>**, **<u>section</u>**, **TAemail, deptName, profEmail**)
- Departments (**<u>name</u>**)
- Professors (**<u>email</u>**, name, **deptName**)

- Take (**<u>studentEmail</u>**, **<u>cNo</u>**, **<u>cSemester</u>**, **<u>cSection</u>**, letterGrade)

Relational Algebra Operators

Relational Data Model

- Structure
  - TABLE or RELATION is the only element
- Value constraints
  - Unique or keys
  - NULLs
- Operations
  - Relational algebra or algebra for tables

TABLE Or Relation

- Schema or table header
  - Attributes or columns
  - Type or domain
    - Primitive: int, float, char[], string or varchar[]
    - Containers not allowed
- A table is seen as a collection (or multiset) of tuples
  - Cannot index in the table

Relational Algebra

- Set of operations or functions on tables
  - Input schema(s) → Output schema
  - Input tuples → Output tuples
- Single table operations
  - Select column, select tuple (row), aggregate, grouping
- Multiple table operations
  - Product and Join, Union, Intersection, Difference

Projection π

- Input table
  - T(A,B,C)
- **A B C**

A	B	C
1	2	3
3	4	6
8	5	4
7	4	3

- **$T' = \pi_{A,\ (A+B+C)\ AS\ S'}(T)$**
- Output table: T'
  - Schema
    - T'(A,S')
  - Same number of tuples as T
  - No duplicate elimination
- **A S'**

A	S'
1	6
3	13
8	17
7	14

Selection σ

- Input table
  - T(A,B,C)
- **A B C**

  1 2 3

  3 4 6

  8 5 4

  7 4 3

- **$T' = \sigma_{A>1 \text{ AND } B+C>A}(T)$**
- Output table: T'
  - Schema
    - T'(A,B,C)
    - Same schema as T
  - Only tuples satisfying predicate
- **A B C**

  3 4 6

  8 5 4

Duplicate Elimination δ

- Input table

  T(A,B)

A	B
0	1
2	3
0	1
2	4
3	4

- **T’ = δ(T)**
- Output table: T’
  - Schema
    - T’(A,B)
    - Same schema as T
  - Only distinct tuples
  - At most the same number of tuples from T

T’(A,B)

A	B
0	1
2	3
2	4
3	4

Sorting τ

- Input table
  T(A,B)

0	1
2	3
0	1
2	4
3	4

- **$T' = \tau_{B\ [DESC]}(T)$**
- Output table: T'
  - Schema
    - T'(A,B)
    - Same schema as T
  - Same tuples sorted

T'(A,B)

2	4
3	4
2	3
0	1
0	1

Aggregations
SUM, AVG, COUNT, MIN, MAX

- Input table

  T(A,B)

0	1
2	3
0	1
2	4
3	4

- **$T' = SUM_A(T)$**
- **$T'' = MAX_{A+B}(T)$**
- Output table: T'
  - Schema
    - T'(X)
  - Single tuple with aggregate result

T'(X)

7

T''(X)

7

GroupBy Aggregations γ

- Input table T(A,B)

0	1
2	3
0	1
2	4
3	4

- $T' = \gamma_{A,\ MIN(B)\ AS\ MB}(T)$
- Output table: T'
  - Schema
    - T'(A, MB)
    - Arguments of γ
  - Tuples have distinct values for A and group aggregate value for other attributes

T'(A,MB)

0	1
2	3
3	4

Set Operations U, ∩, -

- Input tables

R(A,B)		S(A,B)	
1	1	1	2
1	2	4	3
3	4		

- Schema of R, S, and result table T' is the same (A,B)

- Union: T' = R U S

1	1
1	2
3	4
4	3

  - Difference: T' = R - S

1	1
3	4

  - Difference: T' = S - R

4	3

- Intersection: T' = R ∩ S

1	2

Cartesian Product x

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- **T = R x S(A,B)** = {
  (1,1),(1,3),(1,4),
  (1,1),(1,3),(1,4),
  (2,1),(2,3),(2,4),
  (3,1),(3,3),(3,4)}
- The result consists of pairs of one element from R and one from S
- Every element from R is paired with every element from S
- The number of elements in R x S is |R|*|S|, i.e., the size of R multiplied by the size of S

- The schema of the result is the **union** of the R schema and the S schema
  - R(A)
  - S(B)
  - T(A,B) = A U B

Join ⋈

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- **T = R $\bowtie_{A=B}$ S** = {

  (1,1),~~(1,3),(1,4),~~

  (1,1),~~(1,3),(1,4),~~

  ~~(2,1),(2,3),(2,4),~~

  ~~(3,1)~~,**(3,3)**,~~(3,4)~~} = {(1,1),(1,1),(3,3)}
- Join condition between attributes from the two tables
- Only those tuples from the Cartesian product that satisfy the join condition are included in the result

- The schema of the result is the **union** of the R schema and the S schema
  - R(A)
  - S(B)
  - T(A,B) = A U B
- $\mathbf{R \bowtie_{A=B} S = \sigma_{A=B}(RxS)}$

Outer Joins

R(A,B)

0	1
2	3
0	1
2	4
3	4

S(B,C)

0	1
2	4
2	5
3	4
0	2
3	4

$R \bowtie S$ [natural join] (A,B,C)

2	3	4
2	3	4

$R \bowtie_o S$ [full outer join] (A,B,C)

2	3	4
2	3	4
0	1	-
0	1	-
2	4	-
3	4	-
-	0	1
-	2	4
-	2	5
-	0	2

Left (Right) Outer Joins

R(A,B)

0	1
2	3
0	1
2	4
3	4

S(B,C)

0	1
2	4
2	5
3	4
0	2
3	4

$R \bowtie_o S$ [full outer join] (A,B,C)

2	3	4
2	3	4
0	1	-
0	1	-
2	4	-
3	4	-
-	0	1
-	2	4
-	2	5
-	0	2

$R \bowtie_L S$ [left outer join] (A,B,C)

2	3	4
2	3	4
0	1	-
0	1	-
2	4	-
3	4	-

$R \bowtie_R S$ [right outer join] (A,B,C)

2	3	4
2	3	4
-	0	1
-	2	4
-	2	5
-	0	2

Relational Algebra ↔ SQL

- SELECT ↔ Projection π
- FROM ↔ Input tables
- WHERE ↔ Selection σ, Join predicates
- DISTINCT ↔ Duplicate elimination δ
- ORDER BY ↔ Sorting τ
- GROUP BY ↔ GroupBy aggregations γ
- UNION, INTERSECT, EXCEPT ↔ Set operations U, ∩, -
- JOIN ↔ Join

Relational Algebra
Expressions = Queries

Relational Algebra Operators

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U**, ∩, **-**
- Product **x**
- Join **⋈**

- Every operator takes as input one or two tables and generates as output a table
  - Schema
  - Tuples
- Operators are composable
  - The output of one operator is the input of another operator

Relational Algebra Expressions

- Sequence of relational algebra operators
  - Input is a set of tables
  - Output is the result table
- **Relational algebra expression = Query**
- This is exactly how PANDAS work
- Arithmetic algebra mixed operations

  4 * (7 – (2 + 3)) – 6 + 5 * 6

2.4.1 a)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U**, ∩, **-**
- Product **x**
- Join **⋈**

- $S_1(M,S,R,H,P) = \sigma_{S>=3}(PC(M,S,R,H,P))$

  $R(model) = \pi_M(S_1(M,S,R,H,P))$
- $R(model) = \pi_{model}(\sigma_{speed>=3}(PC))$

2.4.1 b)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- $S_1(M,S,R,H,Sc,P) = \sigma_{H>=100}(Laptop(M,S,R,H,Sc,P))$

  $S_2(Ma,M,T,S,R,H,Sc,P) = Product(Ma,M,T) \bowtie S_1(M,S,R,H,Sc,P)$

  $R(maker) = \pi_{Ma}(S_2(Ma,M,T,S,R,H,Sc,P))$

- $R(maker) = \pi_{maker}(Product \bowtie \sigma_{hd>=100}(Laptop))$

2.4.1 c)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- $S_1(model,price) = \pi_{model,price}(\sigma_{maker='B'}(Product) \bowtie PC)$

  $S_2(model,price) = \pi_{model,price}(\sigma_{maker='B'}(Product) \bowtie Laptop)$

  $S_3(model,price) = \pi_{model,price}(\sigma_{maker='B'}(Product) \bowtie Printer)$

  $R(model,price) = S_1 \cup S_2 \cup S_3$

2.4.1 d)

- Projection $\boldsymbol{\pi}$
- Selection $\boldsymbol{\sigma}$
- Duplicate elimination $\boldsymbol{\delta}$
- Sorting $\boldsymbol{\tau}$
- GroupBy aggregations $\boldsymbol{\gamma}$
- Set operations **U**, ∩, **-**
- Product **x**
- Join ⋈

- $S_1(M,C,T,P) = \sigma_{C=true \text{ AND } T='laser'}(Printer(M,C,T,P))$

  $R(model) = \pi_M(S_1(M,C,T,P))$
- $R(model) = \pi_{model}(\sigma_{color=true \text{ AND } type='laser'}(Printer))$

2.4.1 e)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- $S_1(\text{maker}) = \pi_{\text{maker}}(\sigma_{\text{type='laptop'}}(\text{Product}))$

  $S_2(\text{maker}) = \pi_{\text{maker}}(\sigma_{\text{type='pc'}}(\text{Product}))$

  $R(\text{maker}) = S_1 - S_2$

2.4.1 f)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- $S_1(hd,cnt) = \gamma_{hd,\ COUNT(*)\ AS\ cnt}(PC)$

  $S_2(hd,cnt) = \sigma_{cnt>=2}(S_1)$

  $R(hd) = \pi_{hd}(S_2)$
- $R(hd) = \pi_{hd}(\sigma_{cnt>=2}(\gamma_{hd,\ COUNT(*)\ AS\ cnt}(PC)))$

2.4.1 g)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U**, ∩, **-**
- Product **x**
- Join **⋈**

- $S_1(M_1,Sp_1,R_1,H_1,P_1,M_2,Sp_2,R_2,H_2,P_2)$ =
  $PC \rightarrow PC_1(M_1,Sp_1,R_1,H_1,P_1) \bowtie_{Sp1=Sp2 \text{ AND } R1=R2 \text{ AND } M1<M2} PC \rightarrow PC_2(M_2,Sp_2,R_2,H_2,P_2)$

  $R(model_1,model_2)$ =
  $\pi_{M1,M2}(S_1(M_1,Sp_1,R_1,H_1,P_1,M_2,Sp_2,R_2,H_2,P_2))$

2.4.1 h)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- S_1(model,maker) = $\pi_{model,maker}$(Product ⋈ $\sigma_{speed>=2.8}$(PC))

  S_2(model,maker) = $\pi_{model,maker}$(Product ⋈ $\sigma_{speed>=2.8}$(Laptop))

  S_3(model,maker) = $S_1 \cup S_2$

  S_4(maker,cnt) = $\gamma_{maker,\ COUNT(*)\ AS\ cnt}(S_3)$

  S_5(maker,cnt) = $\sigma_{cnt>=2}(S_4)$

  R(maker) = $\pi_{maker}(S_5)$

2.4.1 i)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- $S_1(\text{model,speed}) = \pi_{\text{model,speed}}(\text{PC})$

  $S_2(\text{model,speed}) = \pi_{\text{model,speed}}(\text{Laptop})$

  $S_3(\text{model,speed}) = S_1 \cup S_2$

  $S_4(M_1,Sp_1,M_2,Sp_2) = S_3 \rightarrow S_{31}(M_1,Sp_1) \bowtie_{\text{Sp1<Sp2 AND M1<M2}} S_3 \rightarrow S_{32}(M_2,Sp_2)$

  $S_5(\text{model}) = \pi_{M1}(S_4(M_1,Sp_1,M_2,Sp_2))$

  $S_6(\text{model}) = \pi_{\text{model}}(S_1) \cup \pi_{\text{model}}(S_2)$

  $S_7 = S_6 - S_5$

  $R(\text{maker}) = \pi_{\text{maker}}(\text{Product} \bowtie S_7)$

2.4.1 j)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- $S_1(\text{maker,speed}) = \pi_{\text{maker,speed}}(\text{Product} \bowtie \text{PC})$

  $S_2 = \boldsymbol{\delta}(S_1)$

  $S_3(\text{maker,cnt}) = \gamma_{\text{maker, COUNT(*) AS cnt}}(S_2)$

  $S_4(\text{maker,cnt}) = \sigma_{\text{cnt}>=3}(S_3)$

  $R(\text{maker}) = \pi_{\text{maker}}(S_4)$

2.4.1 k)

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U, ∩, -**
- Product **x**
- Join **⋈**

- $S_1(\text{maker,model}) = \pi_{\text{maker,model}}(\sigma_{\text{type='pc'}}(\text{Product}))$

  $S_2(\text{maker,cnt}) = \gamma_{\text{maker, COUNT(*) AS cnt}}(S_1)$

  $S_3(\text{maker,cnt}) = \sigma_{\text{cnt=3}}(S_2)$

  $R(\text{maker}) = \pi_{\text{maker}}(S_3)$

Relational Algebra
Query Execution Trees

Relational Algebra Operators

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U**, ∩, **-**
- Product **x**
- Join **⋈**

- Every operator takes as input one or two tables and generates as output a table
  - Schema
  - Tuples
- Operators are composable
  - The output of one operator is the input of another operator

Relational Algebra Expressions

- Sequence of relational algebra operators
  - Input is a set of tables
  - Output is the result table
- Relational algebra expression = Query

- $S_1(M,S,R,H,Sc,P) = \sigma_{H>=100}(Laptop(M,S,R,H,Sc,P))$

  $S_2(Ma,M,T,S,R,H,Sc,P) = Product(Ma,M,T) \bowtie S_1(M,S,R,H,Sc,P)$

  $R(maker) = \pi_{Ma}(S_2(Ma,M,T,S,R,H,Sc,P))$
- $R(maker) = \pi_{maker}(Product \bowtie \sigma_{hd>=100}(Laptop))$

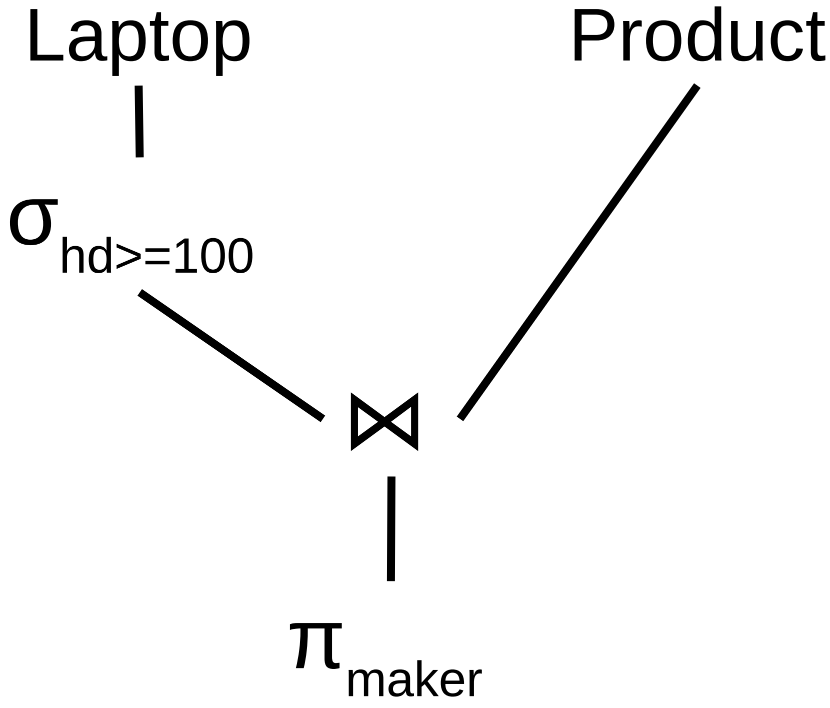
Relational Algebra Expressions ↔ Query Execution Trees

- $S_1(M,S,R,H,Sc,P) = \sigma_{H>=100}(Laptop(M,S,R,H,Sc,P))$

  $S_2(Ma,M,T,S,R,H,Sc,P) = Product(Ma,M,T) \bowtie S_1(M,S,R,H,Sc,P)$

  $R(maker) = \pi_{Ma}(S_2(Ma,M,T,S,R,H,Sc,P))$

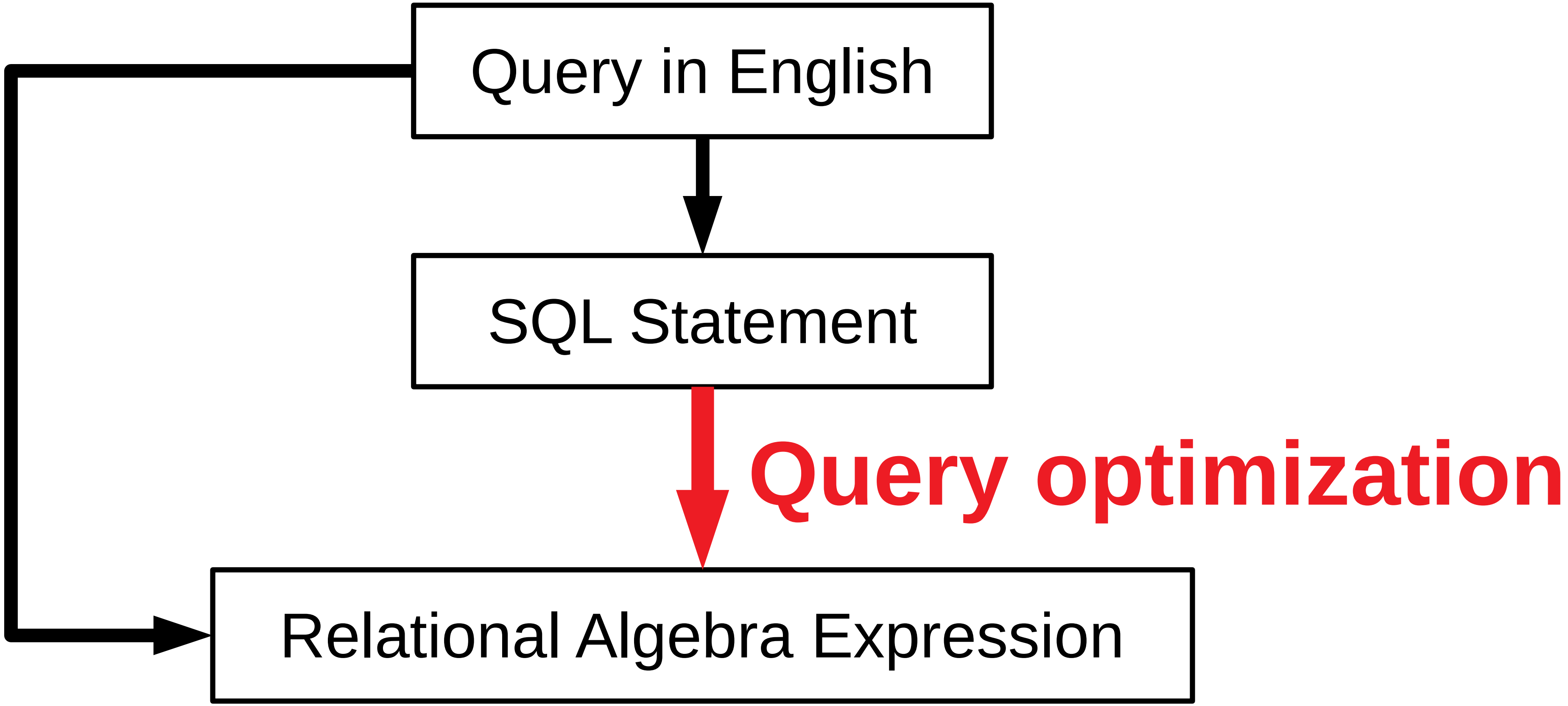
- $R(maker) = \pi_{maker}(Product \bowtie \sigma_{hd>=100}(Laptop))$



Relational Algebra ↔ SQL

- SELECT ↔ Projection π
- FROM ↔ Input tables
- WHERE ↔ Selection σ, Join predicates
- DISTINCT ↔ Duplicate elimination δ
- ORDER BY ↔ Sorting τ
- GROUP BY ↔ GroupBy aggregations γ
- UNION, INTERSECT, EXCEPT ↔ Set operations $\cup$, $\cap$, -
- JOIN ↔ Join

From Queries (Through SQL) To Relational Algebra Expressions



6.1.3 a)

```
select
    model, speed, hd
from pc
where price < 1000
```

PC

$\sigma_{price<1000}$

$\pi_{model,\ speed,\ hd}$

6.1.3 b)

```
select
    model,
    speed as gigahertz,
    hd as gigabytes
from pc
where price < 1000
```

PC

$\sigma_{price<1000}$

$\pi_{model,\ speed\ AS\ gigahertz,\ hd\ AS\ gigabytes}$

6.1.3 c)

```sql
select distinct maker
from product
where type = 'printer'
```

Product

|

$\sigma_{type='printer'}$

|

π_{maker}

|

δ

6.1.3 d)

```
select
    model, ram, screen
from laptop
where price > 1500
```

Laptop

$\sigma_{price>1500}$

$\pi_{model,\ ram,\ screen}$

6.1.3 e)

```
select *
from printer
where color = true
```

Printer

$\sigma_{color=true}$

6.1.3 f)

select model, hd

from pc

where speed = 3.2
and price < 2000

PC

|

$\sigma_{speed=3.2 \text{ AND } price<2000}$

|

$\pi_{model,\ hd}$

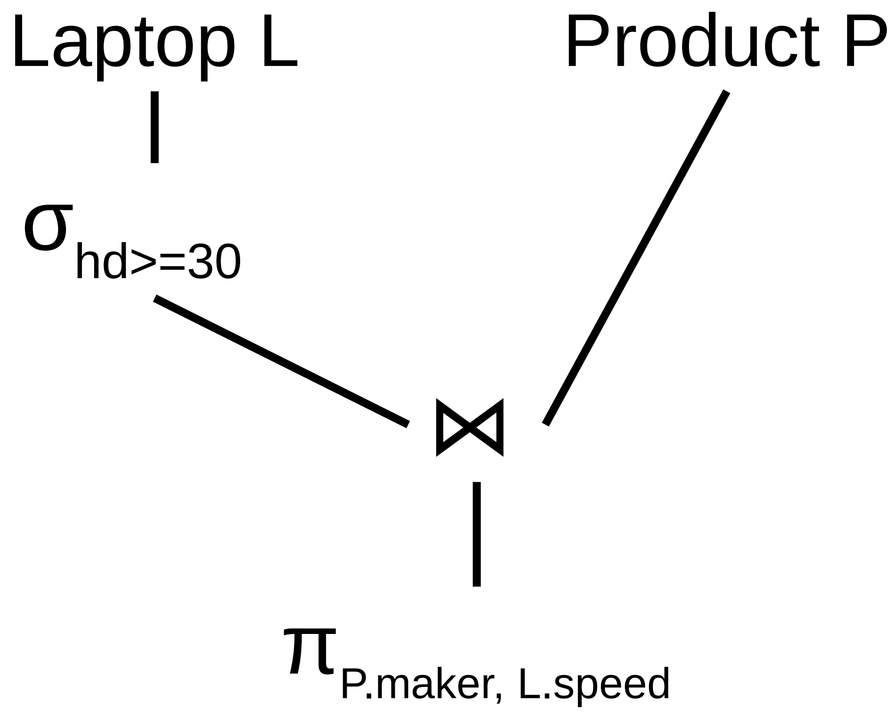
6.2.2 a)

select P.maker, L.speed

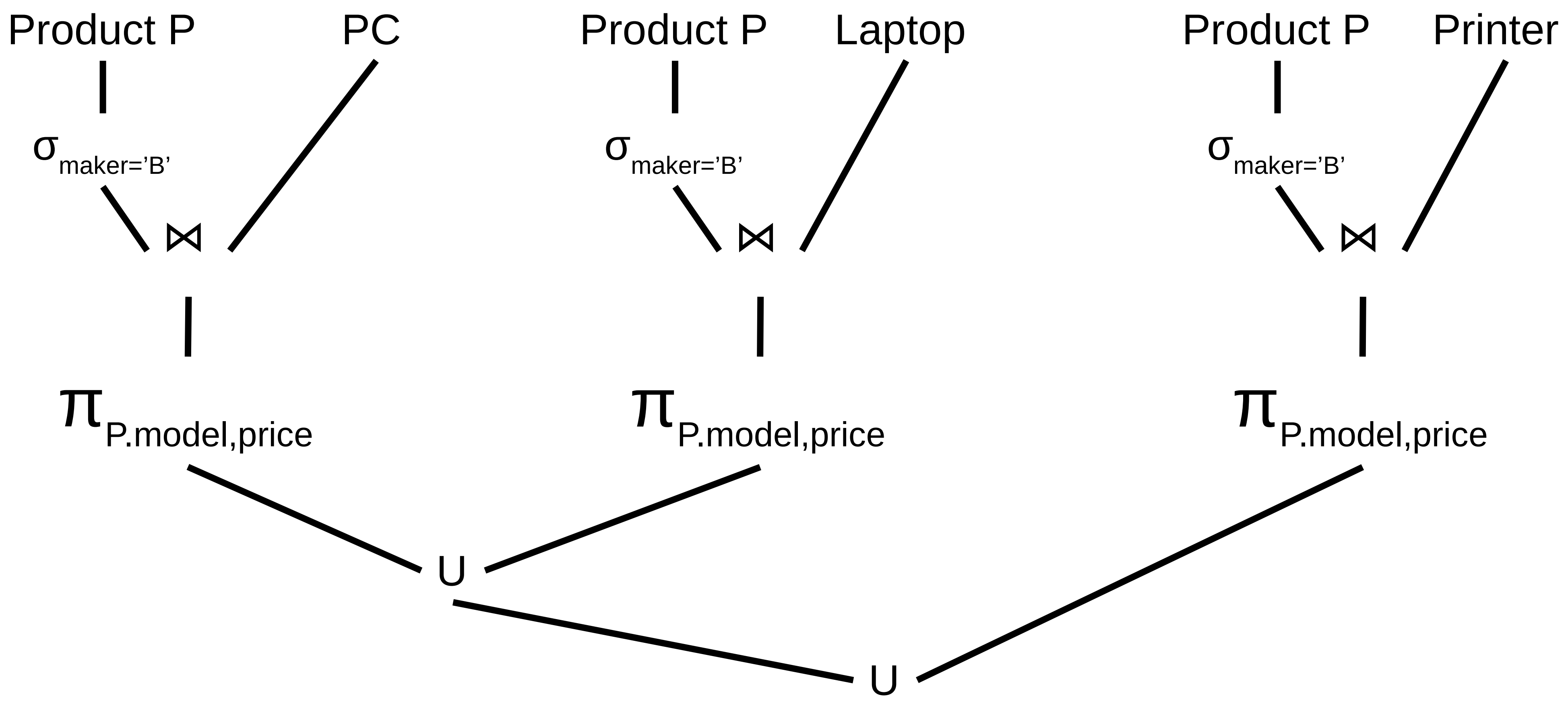
from Product P, Laptop L

where P.model = L.model

AND hd >= 30

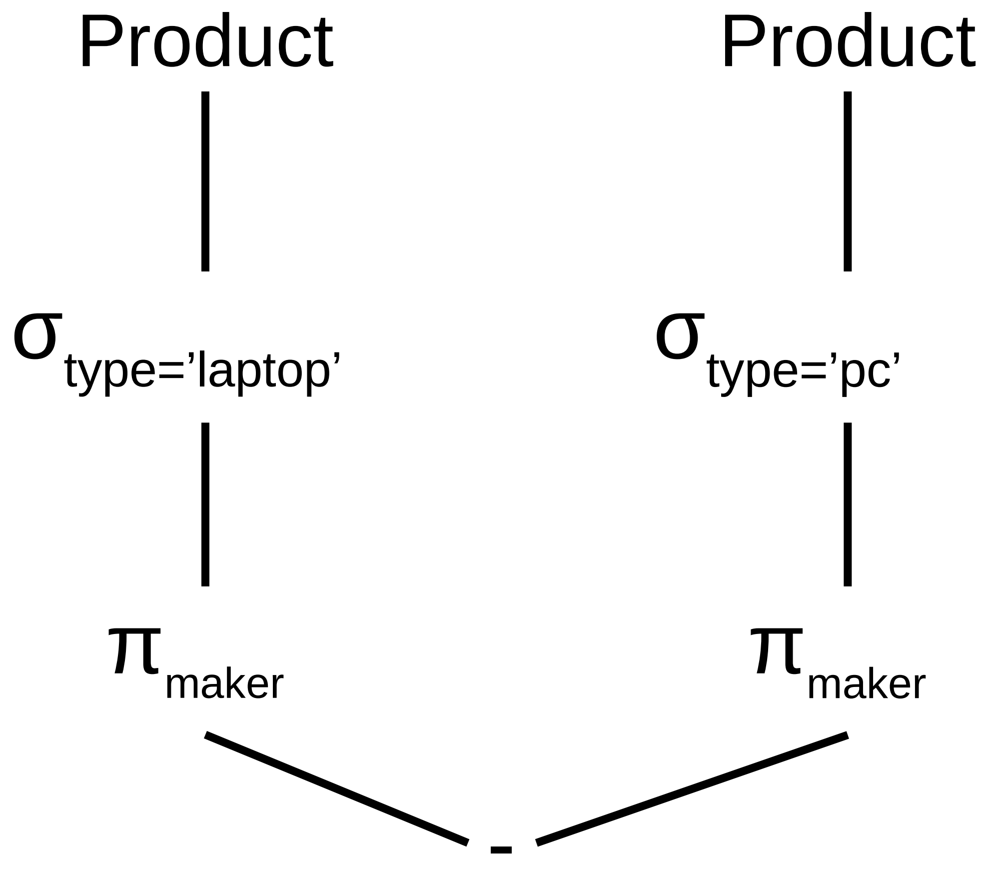


6.2.2 b)


Product P
PC
Product P
Laptop
Product P
Printer
$\sigma_{maker='B'}$
$\sigma_{maker='B'}$
$\sigma_{maker='B'}$
⋈
⋈
⋈
$\pi_{P.model,price}$
$\pi_{P.model,price}$
$\pi_{P.model,price}$
U
U

6.2.2 c)

```
select maker
from Product
where type = 'laptop'
EXCEPT
select maker
from Product
where type = 'pc'
```



6.2.2 d)

select distinct PC1.hd

from PC PC1, PC PC2

where PC1.hd = PC2.hd AND

PC1.model > PC2.model

PC PC1 PC PC2

$\bowtie_{\text{PC1.hd=PC2.hd AND PC1.model>PC2.model}}$

$\pi_{\text{PC1.hd}}$

δ

6.2.2 e)

select PC1.model as model_1,
PC2.model as model_2

from PC PC1, PC PC2

where PC1.speed = PC2.speed
AND PC1.ram = PC2.ram

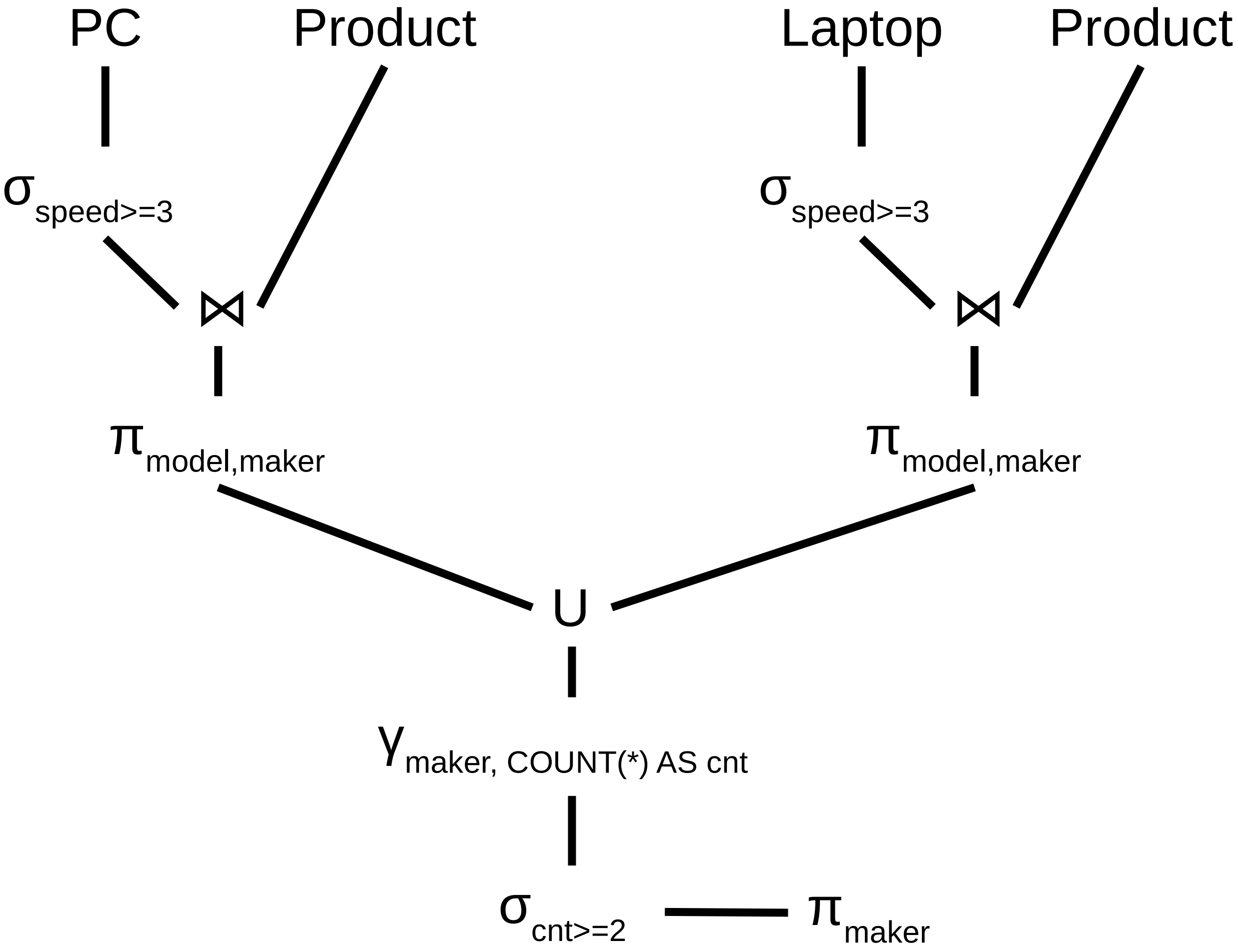
AND PC1.model < PC2.model

PC PC1 PC PC2

$\bowtie_{\text{PC1.speed=PC2.speed AND PC.ram=PC2.ram AND PC1.model<PC2.model}}$

$\pi_{\text{PC1.model AS model\_1, PC2.model AS model\_2}}$

6.2.2 f)



- $S_1(model,maker) = \pi_{model,maker}(Product \bowtie \sigma_{speed>=3}(PC))$

  $S_2(model,maker) = \pi_{model,maker}(Product \bowtie \sigma_{speed>=3}(Laptop))$

  $S_3(model,maker) = S_1 \cup S_2$
  $S_4(maker,cnt) = \gamma_{maker,\ COUNT(*)\ AS\ cnt}(S_3)$

  $S_5(maker,cnt) = \sigma_{cnt>=2}(S_4)$

  $R(maker) = \pi_{maker}(S_5)$

Relational Algebra
Query Execution Tree Examples

Relational Algebra Operators

- Projection **π**
- Selection **σ**
- Duplicate elimination **δ**
- Sorting **τ**
- GroupBy aggregations **γ**
- Set operations **U**, **∩**, **-**
- Product **x**
- Join **⋈**

- Every operator takes as input one or two tables and generates as output a table
  - Schema
  - Tuples
- Operators are composable
  - The output of one operator is the input of another operator

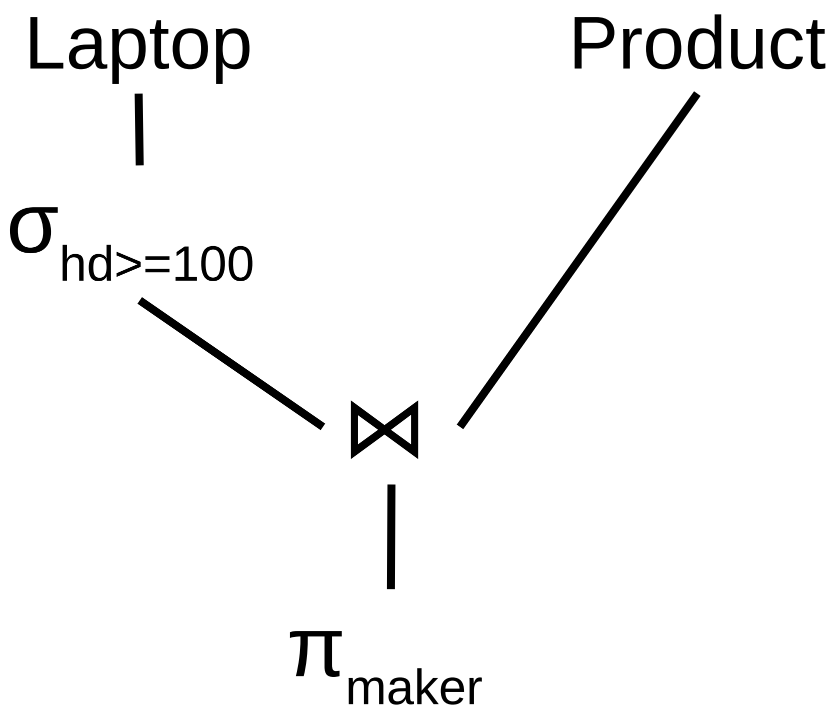
Relational Algebra Expressions ↔ Query Execution Trees

- $S_1(M,S,R,H,Sc,P) = \sigma_{H>=100}(Laptop(M,S,R,H,Sc,P))$

  $S_2(Ma,M,T,S,R,H,Sc,P) = Product(Ma,M,T) \bowtie S_1(M,S,R,H,Sc,P)$

  $R(maker) = \pi_{Ma}(S_2(Ma,M,T,S,R,H,Sc,P))$

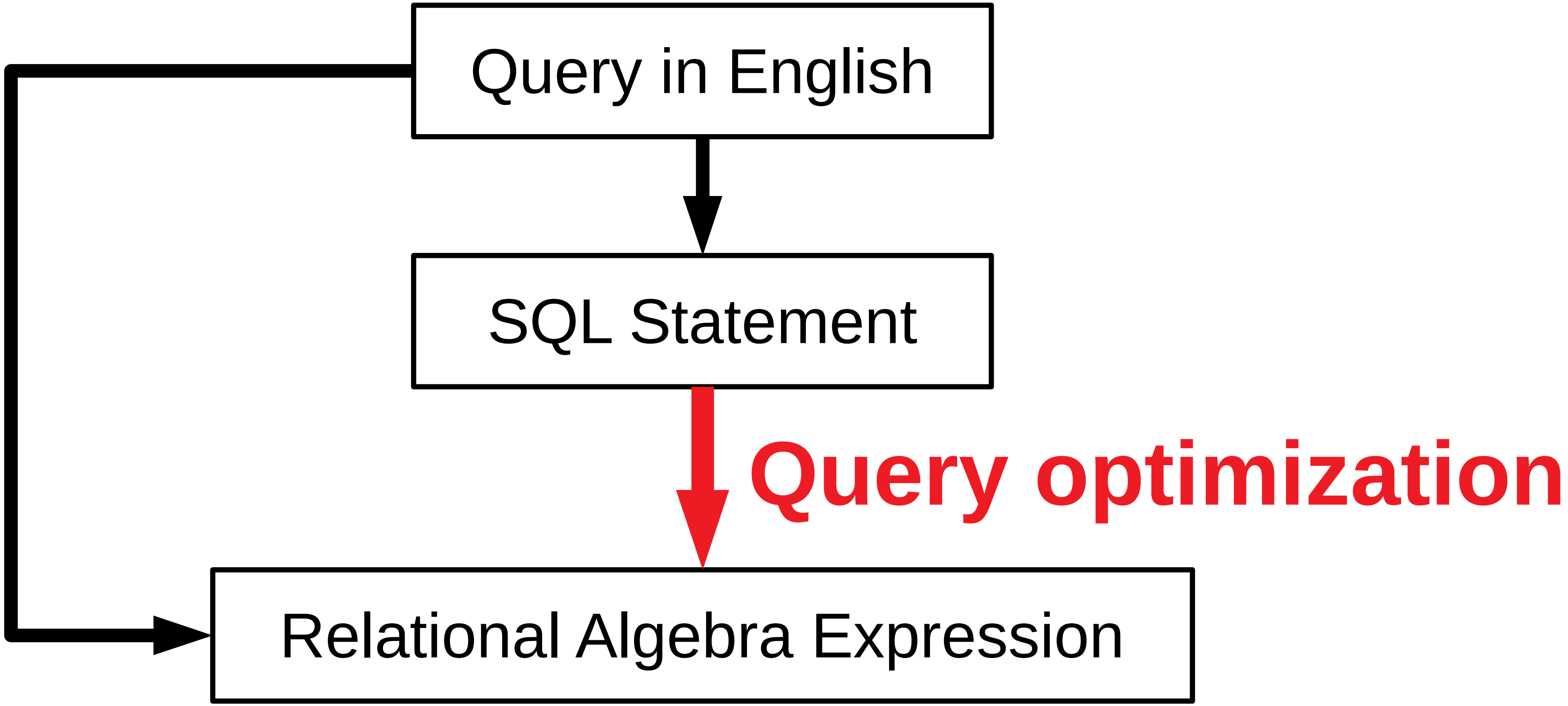
- $R(maker) = \pi_{maker}(Product \bowtie \sigma_{hd>=100}(Laptop))$



Relational Algebra ↔ SQL

- SELECT ↔ Projection π
- FROM ↔ Input tables
- WHERE ↔ Selection σ, Join predicates
- DISTINCT ↔ Duplicate elimination δ
- ORDER BY ↔ Sorting τ
- GROUP BY ↔ GroupBy aggregations γ
- UNION, INTERSECT, EXCEPT ↔ Set operations $\cup$, $\cap$, -
- JOIN ↔ Join

From Queries (Through SQL) To Relational Algebra Expressions



6.3.1 a)

select maker

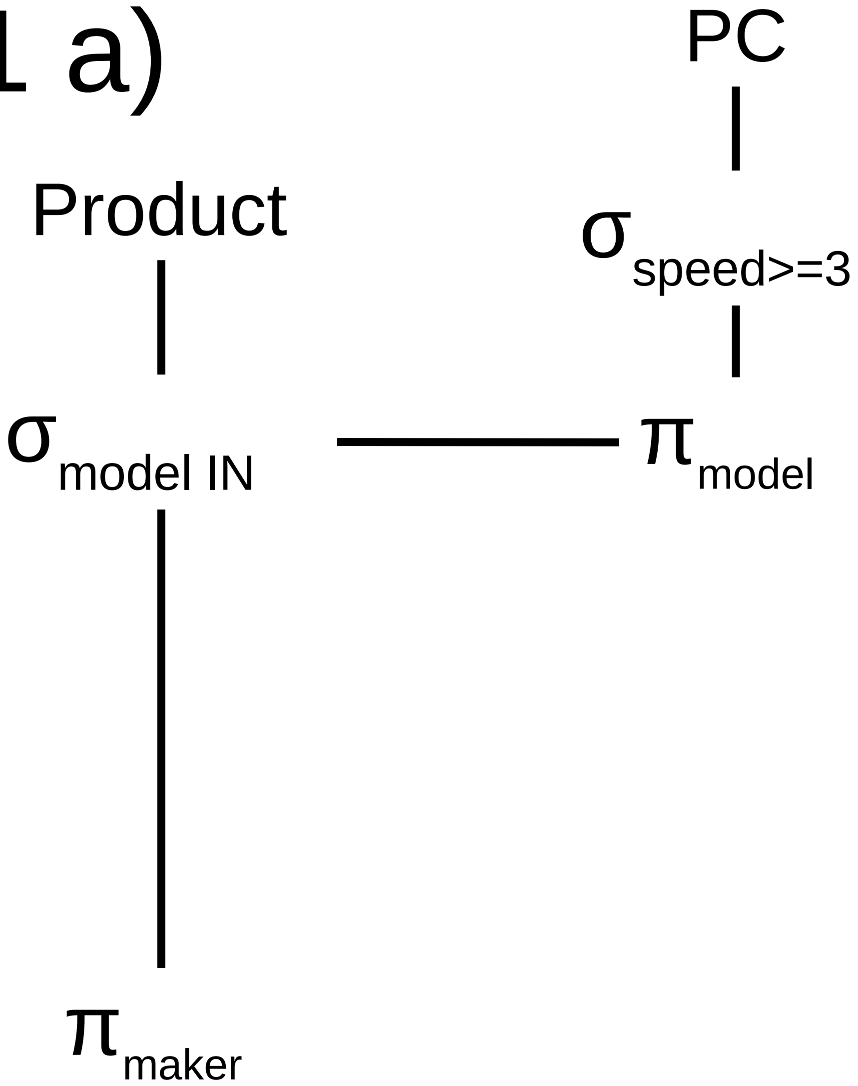
from Product

where model in

(select model

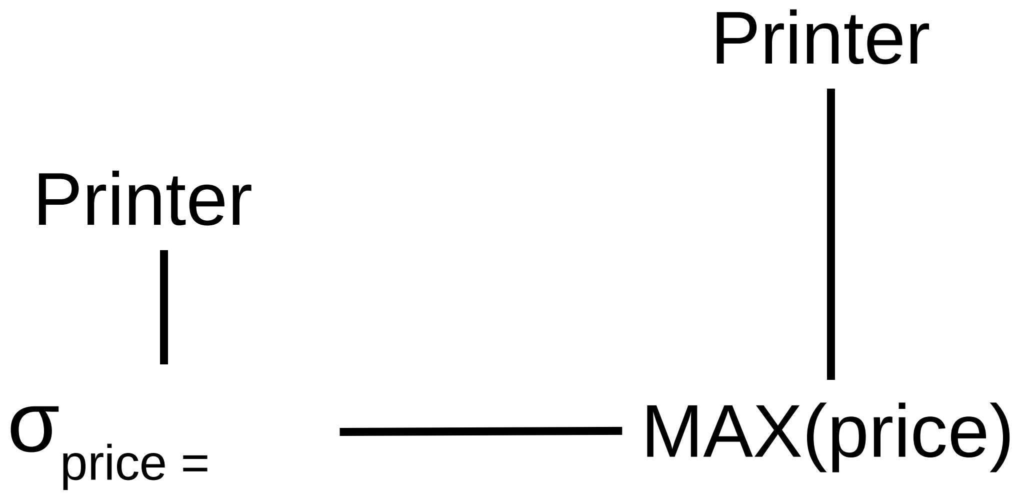
from PC

where speed >= 3)



6.3.1 b)

```
select *
from Printer
where price =
      (select max(price)
       from Printer)
```

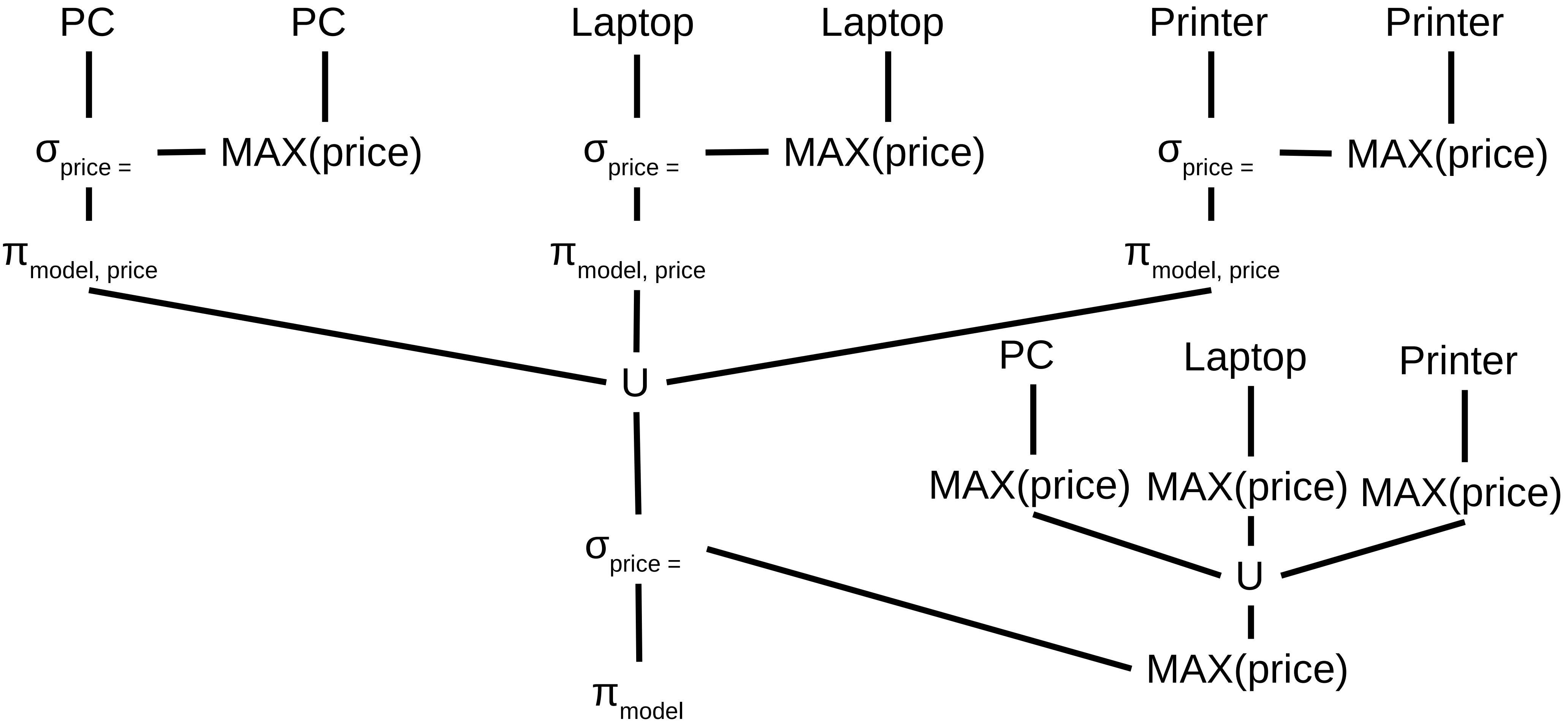


6.3.1 c)

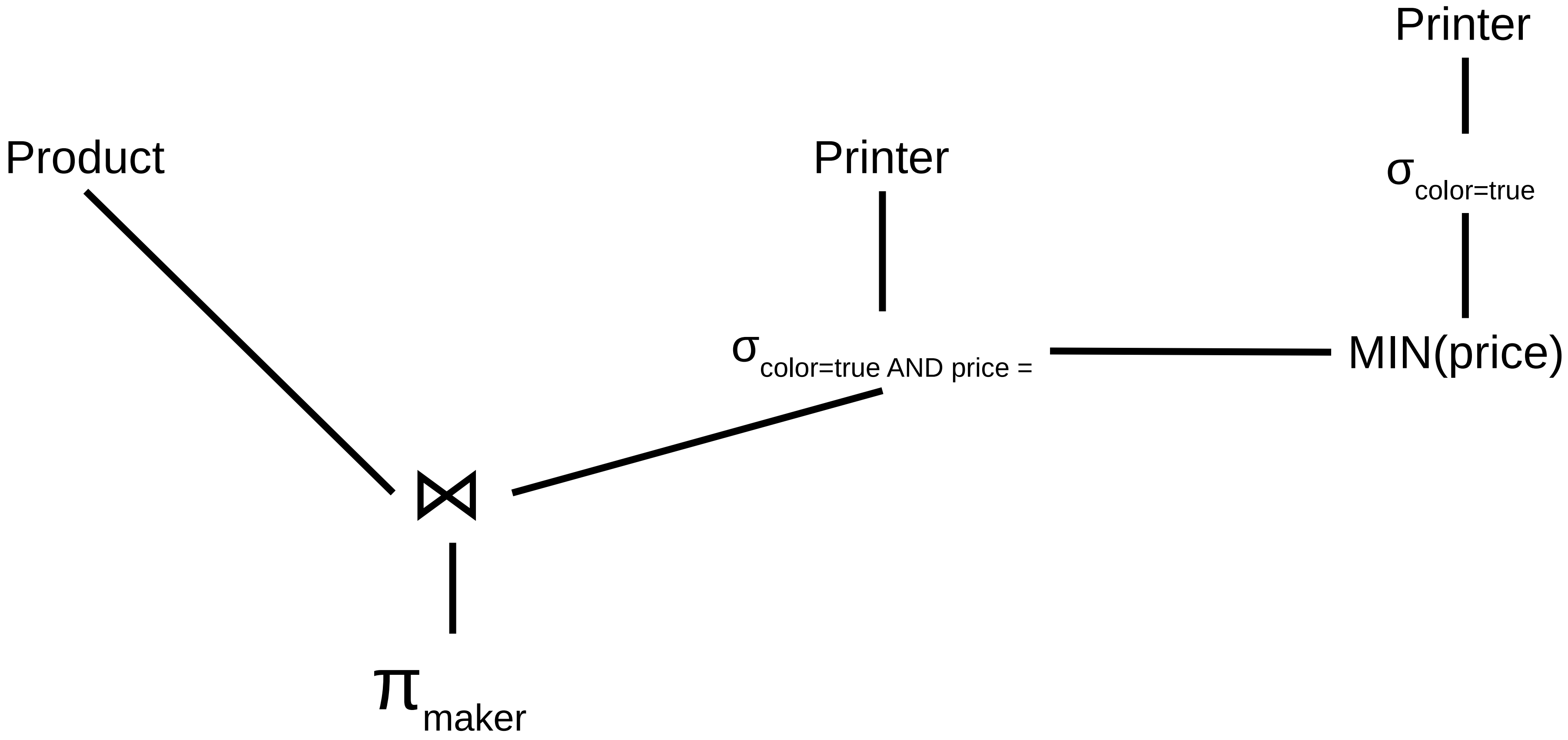
```
select *
from Laptop L
where not exists
      (select *
       from PC
       where PC.speed < L.speed)
```

$$
\begin{array}{ccc}
\text{Laptop L} & & \text{PC} \\
| & & | \\
\sigma_{\text{NOT EXISTS}} & — & \sigma_{\text{PC.speed<L.speed}}
\end{array}
$$

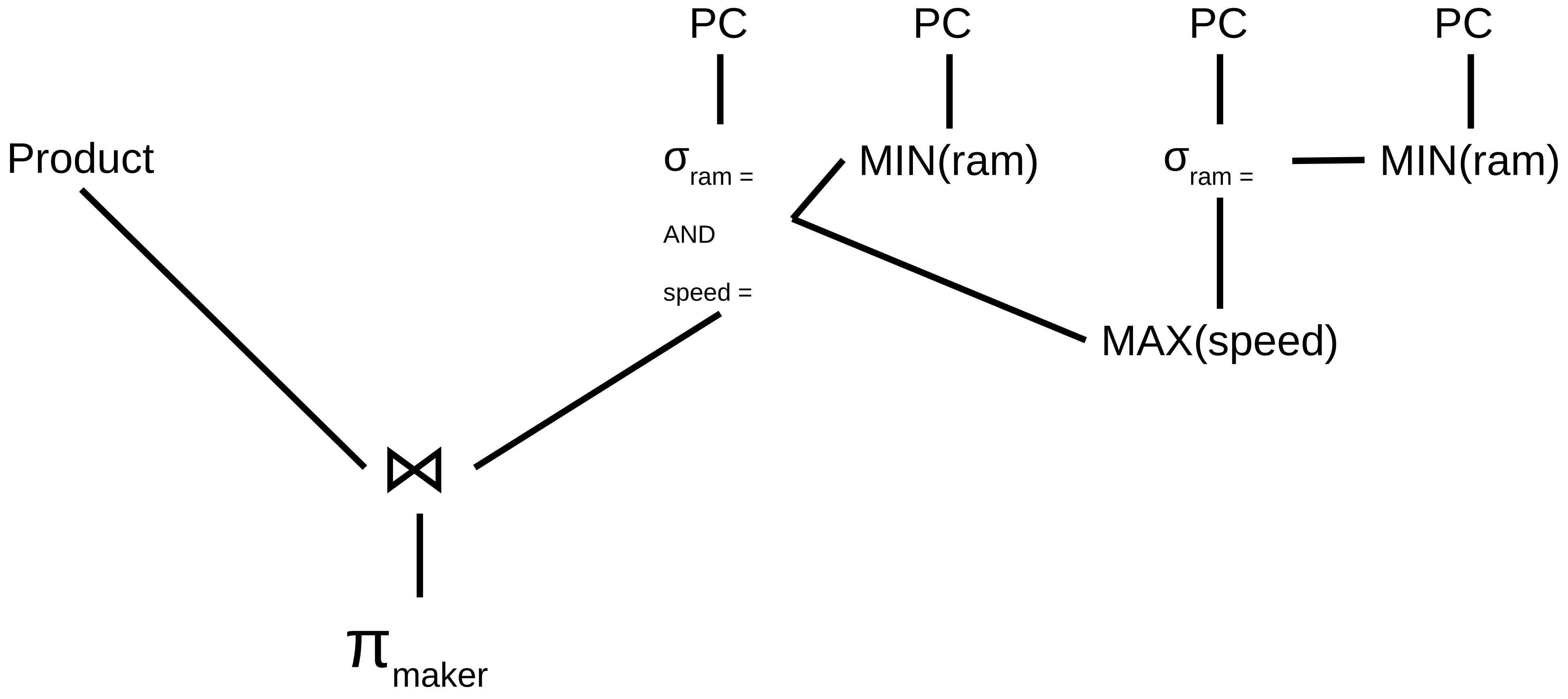
6.3.1 d)


PC
PC
Laptop
Laptop
Printer
Printer
σprice =
MAX(price)
σprice =
MAX(price)
σprice =
MAX(price)
πmodel, price
πmodel, price
πmodel, price
PC
Laptop
Printer
U
MAX(price)
MAX(price)
MAX(price)
σprice =
U
πmodel
MAX(price)

6.3.1 e)


Printer
σ color=true
MIN(price)
Product
Printer
σ color=true AND price =
⋈
π maker

6.3.1 f)


PC
PC
PC
PC
Product
σ ram = AND speed =
MIN(ram)
σ ram =
MIN(ram)
MAX(speed)
⋈
π maker

6.4.6 a)

select avg(speed) as avg_speed

from pc

PC

AVG(speed) AS avg_speed

6.4.6 b)

select avg(speed) as avg_speed

from laptop

where price > 1000

Laptop

$\sigma_{price>1000}$

AVG(speed) AS avg_speed

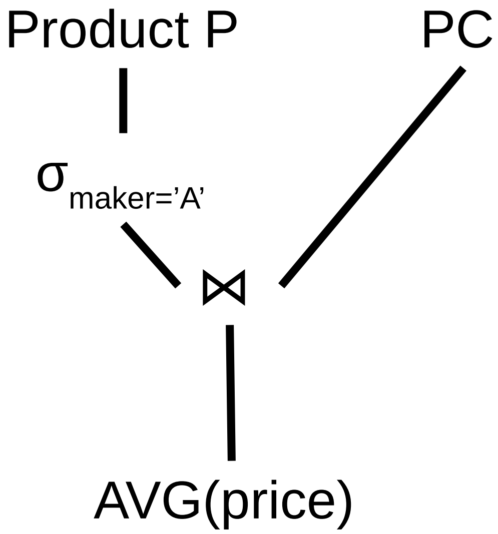
6.4.6 c)

select avg(price)

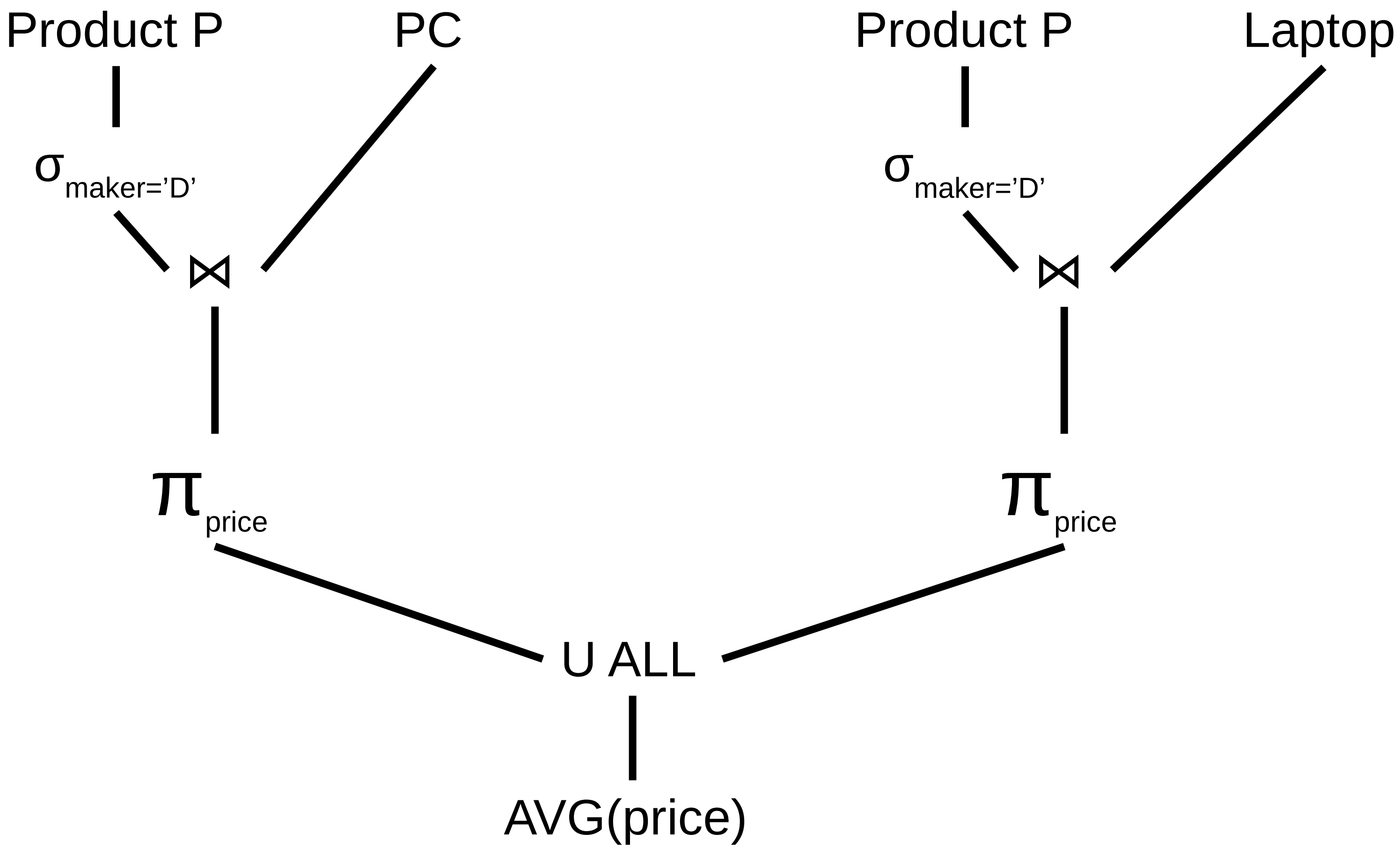
from Product P, PC

where P.model = PC.model AND

P.maker = 'A'



6.4.6 d)


Product P
PC
Product P
Laptop
σ maker='D'
σ maker='D'
⋈
⋈
π price
π price
U ALL
AVG(price)

6.4.6 e)

select speed, avg(price) as avg_price

from pc

group by speed

PC

|

$\gamma_{\text{speed, AVG(price) AS avg\_price}}$

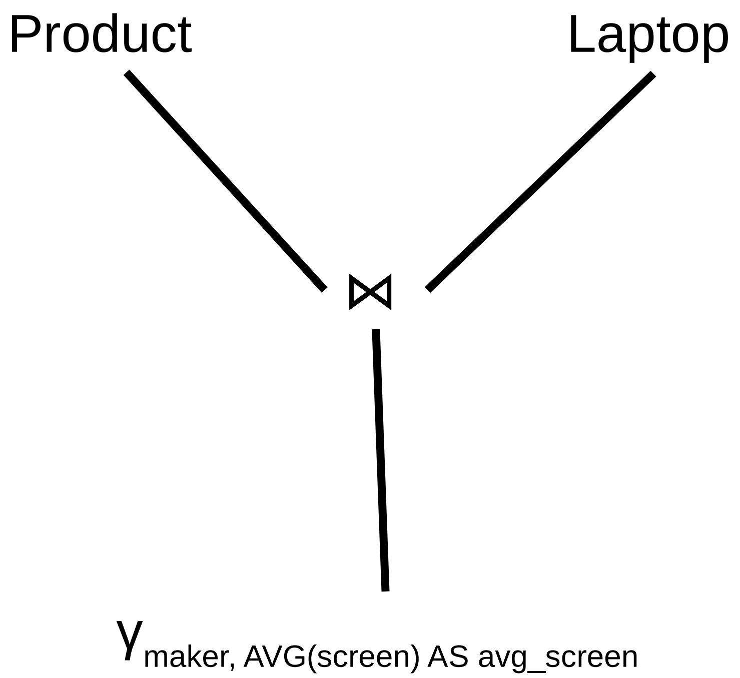
6.4.6 f)

select maker, avg(screen) as avg_screen

from Product P, Laptop L

where P.model = L.model

group by maker



6.4.6 g)

select maker, count (distinct model) as models

from product

where type = 'pc'

group by maker

having models >= 3

Product

$\sigma_{type='pc'}$

$\gamma_{maker,\ COUNT(DISTINCT\ model)\ AS\ models}$

$\sigma_{models>=3}$

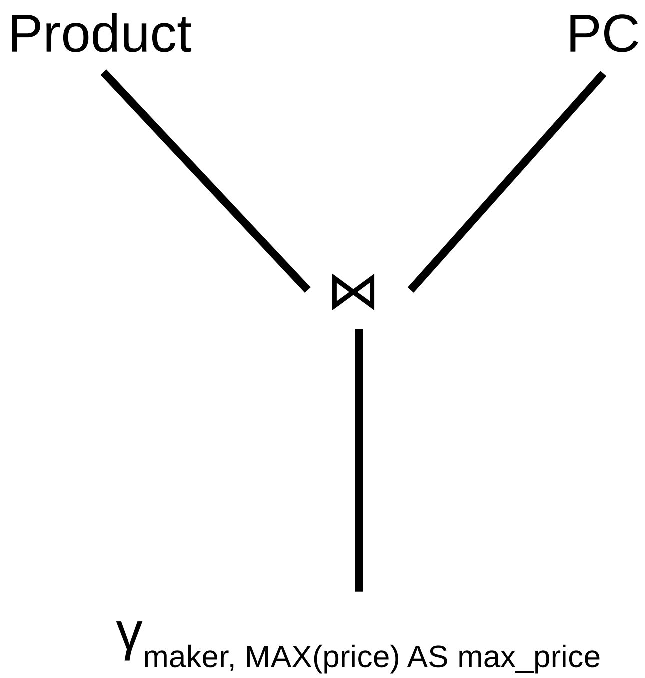
6.4.6 h)

select maker, max(price) as max_price

from Product P, PC

where P.model = PC.model

group by maker



6.4.6 i)

select speed, avg(price)
as avg_price

from pc

where speed > 2

group by speed

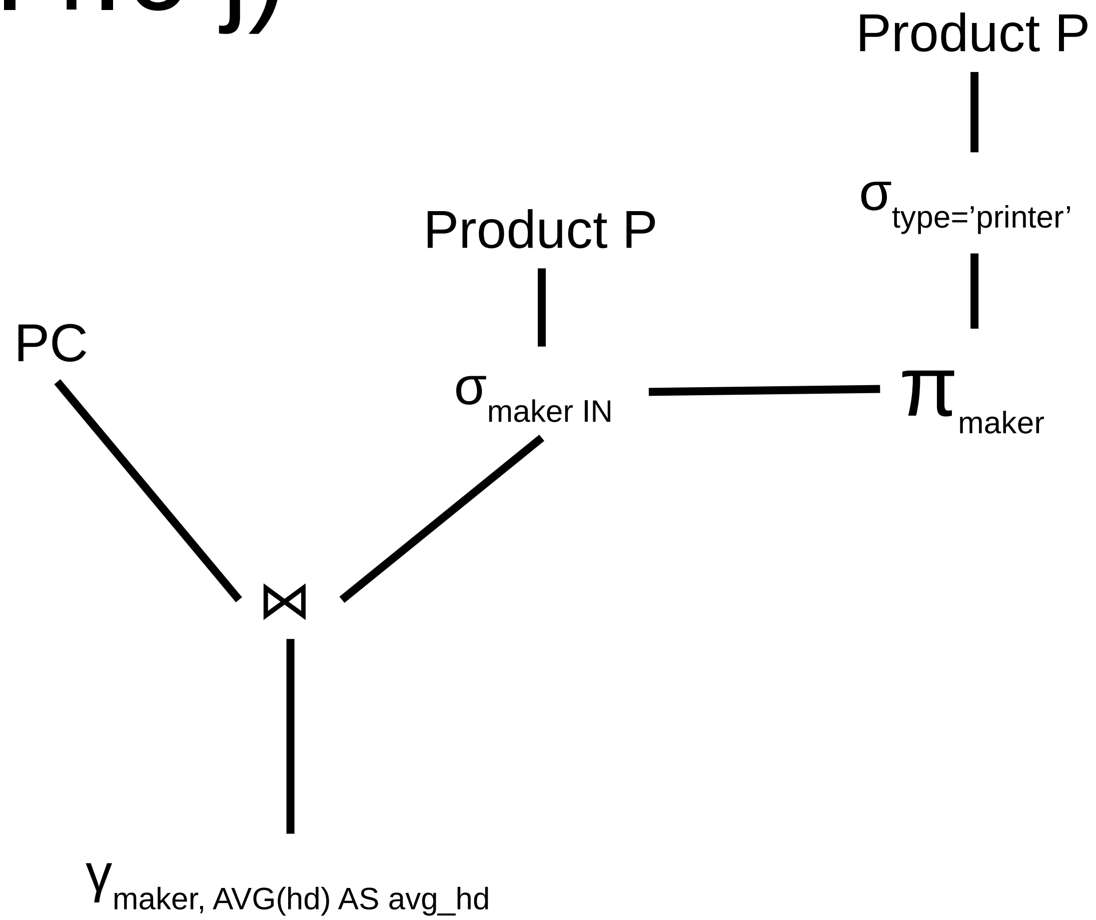
PC

$\sigma_{speed>2}$

$\gamma_{speed,\ AVG(price)\ AS\ avg\_price}$

6.4.6 j)

```
select maker, avg(hd) as avg_hd
from Product P, PC
where P.model = PC.model AND
    maker in (select maker
              from Product
              where type = 'printer')
group by maker
```



Database Application Development
Java JDBC

Packages

- Install Java JDK
  - Ubuntu: package *openjdk-11-jdk*
- Install *Java Extension Pack* in VSCode
  - Automatically detects installed Java JDK
- Download SQLite JDBC driver
  - https://github.com/xerial/sqlite-jdbc
  - Read instructions carefully
  - Add jar to Java classpath

JDBC Tutorials

- SQLite
  - https://www.tutorialspoint.com/sqlite/sqlite_java.htm
- MySQL
  - https://www.tutorialspoint.com/jdbc/index.htm

Database Application Development
Python SQLite3

Packages

- Install Python3
  - Ubuntu: package *python3*
- Install *Python Extension Pack* in VSCode
  - Automatically detects installed Python 3.7
- Install SQLite module for Python
  - https://stackoverflow.com/questions/19530974/how-can-i-add-the-sqlite3-module-to-python
  - Read instructions carefully

Python SQLite Documentation

- https://docs.python.org/3/library/sqlite3.html
- https://pythonexamples.org/python-sqlite3-tutorial/

Database Web Application Development
Apache + PHP

Packages

- Install Apache HTTP Server, PHP, and PHP-SQLite
  - Ubuntu packages: *apache2 php7.2 php7.2-sqlite3*
- Install *PHP Extension Pack* in VSCode
  - Automatically detects installed PHP
- Activate sqlite3 extension in *php.ini*
  - */etc/php/7.2/apache2/php.ini*
  - Uncomment line with sqlite3

Tutorials

- Install and configure Apache2
  - https://dzone.com/articles/how-to-install-and-configure-apache2
- PHP webpage design and implementation
  - https://www.itdominator.com/php7-sqlite3-ajax-tutorial/
- SQLite in PHP
  - https://www.tutorialspoint.com/sqlite/sqlite_php.htm

Database Web Application Development
Node.js + JavaScript

Packages

- Install Node.js language and npm package manager
  - Ubuntu packages: *nodejs, npm*
- Add *sqlite3* and *express* extensions to Node.js project
  - *npm install sqlite3*
  - *npm install express*

Tutorials

- Install and configure Node.js
  - https://itsfoss.com/install-nodejs-ubuntu/
- Rest API in Node.js
  - https://developerhowto.com/2018/12/29/build-a-rest-api-with-node-js-and-express-js/
- SQLite in Node.js
  - https://stackabuse.com/a-sqlite-tutorial-with-node-js/#disqus_thread
- Access Rest API from JavaScript client
  - https://rapidapi.com/blog/how-to-use-an-api-with-javascript/

SQL Injection

SQL Injection

- Application does not handle user input securely
- User provides input that changes behavior of SQL statement
  - Extract additional data beyond what is expected
  - Perform malicious modification operations on databases
    - Insert invalid tuples
    - Delete complete tables
- **SOLUTION: ALWAYS USE PREPARED STATEMENTS**

Python Application Code

- Insecure
  - def printerByType_insecure(_conn, _type):

```
sql = """select model, price
    from Printer
    where type = '{}'""".format(_type)
```

- Secure (prepared)
  - def printerByType_secure(_conn, _type):

```
sql = """select model, price
    from Printer
    where type = ?"""
args = [_type]
```

Print the Full Table Content

- sql = """select model, price

  from Printer

  where type = **'{}'**""".format(_type)
- printerByType_insecure(conn, "laser")
- printerByType_insecure(conn, "laser**' OR '1'='1**")

Extract Attribute Values (Extra Tuples)

- sql = """select model, price

  from Printer

  where type = **'{}'**""".format(_type)
- printerByType_insecure(conn, "laser**' OR type LIKE \'%ink%**")
- printerByType_insecure(conn,

  """laser**' UNION**

  select model, price from PC --""")

Extract Attribute Names

- sql = """select model, price

  from Printer

  where type = **'{}'**""".format(_type)
- printerByType_insecure(conn, "laser**' AND color = true --**")
- printerByType_insecure(conn,

  """laser**' UNION**

  select name, sql from sqlite_master where type = 'table'--""")

Extract Table Names

- sql = """select model, price

  from Printer

  where type = **'{}'**""".format(_type)
- printerByType_insecure(conn,

  """laser**' AND 13 = (select count(*) from PC) --**""")
- printerByType_insecure(conn,

  """laser**' UNION**

  select name, tbl_name from sqlite_master where type = 'table'--""")

Perform Modification Operations

- sql = """select model, price
  from Printer
  where type = **'{}'**""".format(_type)
- execute(sql)
- printerByType_insecure(conn,
  """laser**'; insert into printer (price) values(300); --**""")
- **executescript(sql)**
- printerByType_**script**_insecure(conn,
  """laser**'; insert into printer (price) values(300); --**""")