

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

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

```
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

- $A \cup B = \{1,2,3,5\}$

- Intersection

- $A \cap B = \{1,3\}$

- Difference

- $A - B = \{2\}$

- $B - A = \{5\}$

- Multi-sets or bags

- $A' = \{1,1,2,3,3\}$

- $B' = \{1,2,2,3,4\}$

- Union

- $A' \cup B' = \{1,1,1,2,2,2,3,3,3,4\}$

- Intersection

- $A' \cap B' = \{1,2,3\}$

- 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 **table₁, table₂**
[WHERE table_predicates AND **join_conditions**]
[GROUP BY grouping_attributes
 [HAVING agg_condition]]
[ORDER BY sorting_attributes]
[UNION [ALL]] [INTERSECT] [EXCEPT]

Cartesian Product

- $R(A) = \{1,1,2,3\}$
- $S(B) = \{1,3,4\}$
- $R \times S(A,B) = \{$
 $(1,1),(1,3),(1,4),$
 $(1,1),(1,3),(1,4),$
 $(2,1),(2,3),(2,4),$
 $(3,1),(3,3),(3,4)\}$
- 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 $R \times S$ is $|R| \times |S|$, 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

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

Two-Table Join

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

$$\begin{aligned}
 & \bullet R \bowtie_{A=B} S \bowtie_{B>C} T(A,B,C) = \{ \\
 & \quad (\underline{1}, \underline{1}, \underline{2}), (\underline{1}, \underline{3}, \underline{2}), (\underline{1}, \underline{4}, \underline{2}), \\
 & \quad (\underline{1}, \underline{1}, \underline{2}), (\underline{1}, \underline{3}, \underline{2}), (\underline{1}, \underline{4}, \underline{2}), \\
 & \quad (\underline{2}, \underline{1}, \underline{2}), (\underline{2}, \underline{3}, \underline{2}), (\underline{2}, \underline{4}, \underline{2}), \\
 & \quad (\underline{3}, \underline{1}, \underline{2}), (\underline{3}, \underline{3}, \underline{2}), (\underline{3}, \underline{4}, \underline{2}), \\
 & \quad (\underline{1}, \underline{1}, \underline{4}), (\underline{1}, \underline{3}, \underline{4}), (\underline{1}, \underline{4}, \underline{4}), \\
 & \quad (\underline{1}, \underline{1}, \underline{4}), (\underline{1}, \underline{3}, \underline{4}), (\underline{1}, \underline{4}, \underline{4}), \\
 & \quad (\underline{2}, \underline{1}, \underline{4}), (\underline{2}, \underline{3}, \underline{4}), (\underline{2}, \underline{4}, \underline{4}), \\
 & \quad (\underline{3}, \underline{1}, \underline{4}), (\underline{3}, \underline{3}, \underline{4}), (\underline{3}, \underline{4}, \underline{4}) \} = \{(\underline{3}, \underline{3}, \underline{2})\}
 \end{aligned}$$

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 table₁, table₂, ...

[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)	S(B,C)	R ⋈ S [natural join] (A,B,C)	R ⋈ S [full outer join] (A,B,C)
0 1	0 1	2 3 4	2 3 4
2 3	2 4	2 3 4	0 1 -
0 1	2 5	2 3 4	0 1 -
2 4	3 4		2 4 -
2 4	0 2		3 4 -
3 4	3 4		- 0 1
			- 2 4
			- 2 5
			- 0 2

Left (Right) Outer Joins

R(A,B)		S(B,C)		R ⋈ S [full outer join] (A,B,C)			R ⋈ _L S [left outer join] (A,B,C)			R ⋈ _R S [right outer join] (A,B,C)		
0	1	0	1	2	3	4	2	3	4	2	3	4
2	3	2	4	0	1	-	2	3	4	2	3	4
0	1	2	5	0	1	-	2	3	4	2	3	4
2	4	3	4	2	4	-	0	1	-	-	0	1
3	4	0	2	3	4	-	0	1	-	-	2	4
		3	4	-	0	1	2	4	-	-	2	5
				-	2	5	3	4	-	-	0	2
				-	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**

**)**

# 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
- 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)

**LIMIT 1**

# 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**

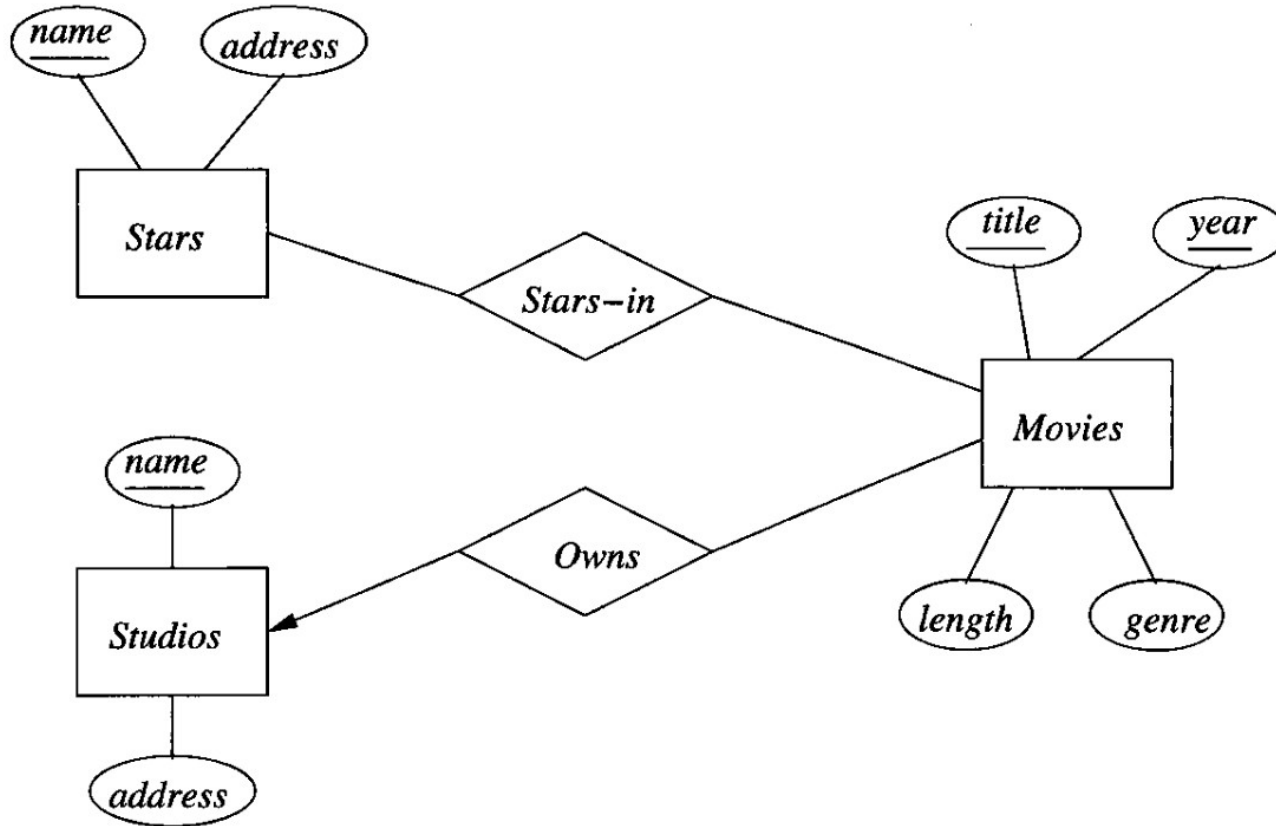
# Examples

- Computers
- TPCH

# E/R Diagrams

## Mapping to Relations

# E/R to Relations



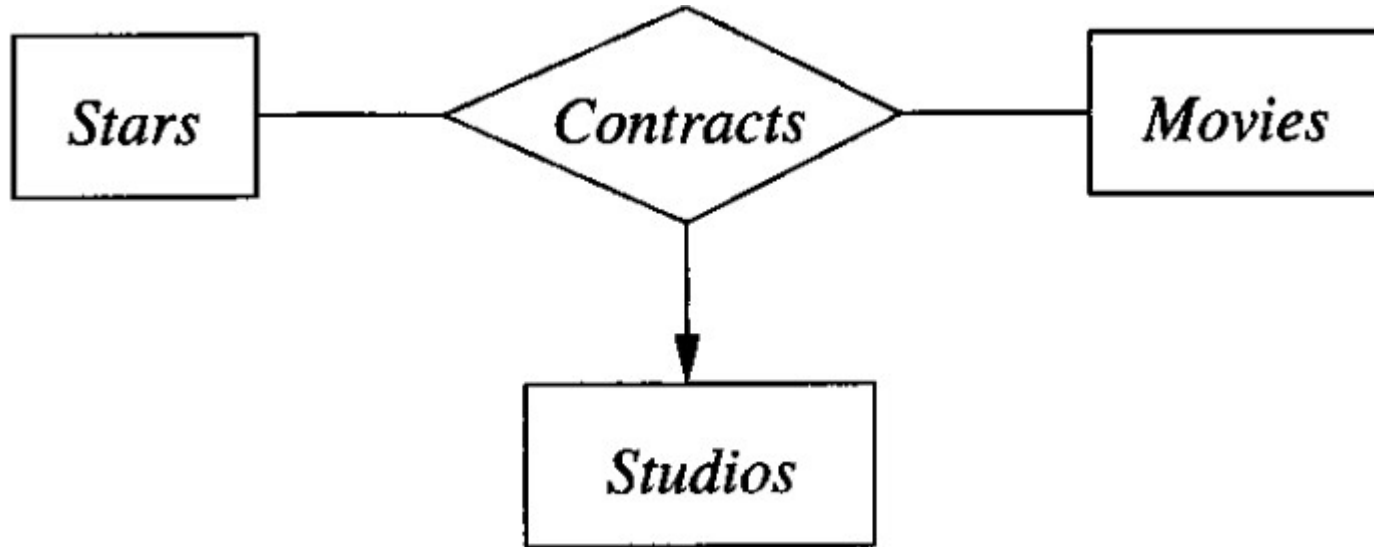
- Entities
  - Stars (name, address)
  - Movies (title, year, length, genre, **studioName**)
  - Studios (name, address)
- Many-to-many relationships
  - Stars-in (starName, movieTitle, movieYear)

# One-to-one (-many) Relationships



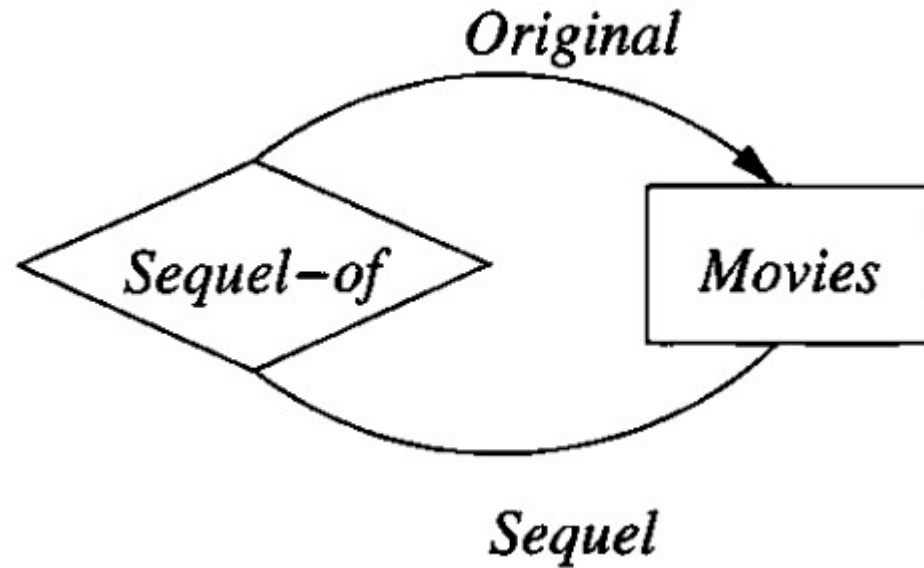
- Studios (name, address, **presidentName**)
- Presidents (name, **studioName**)

# Multi (Three)-Way Relationships



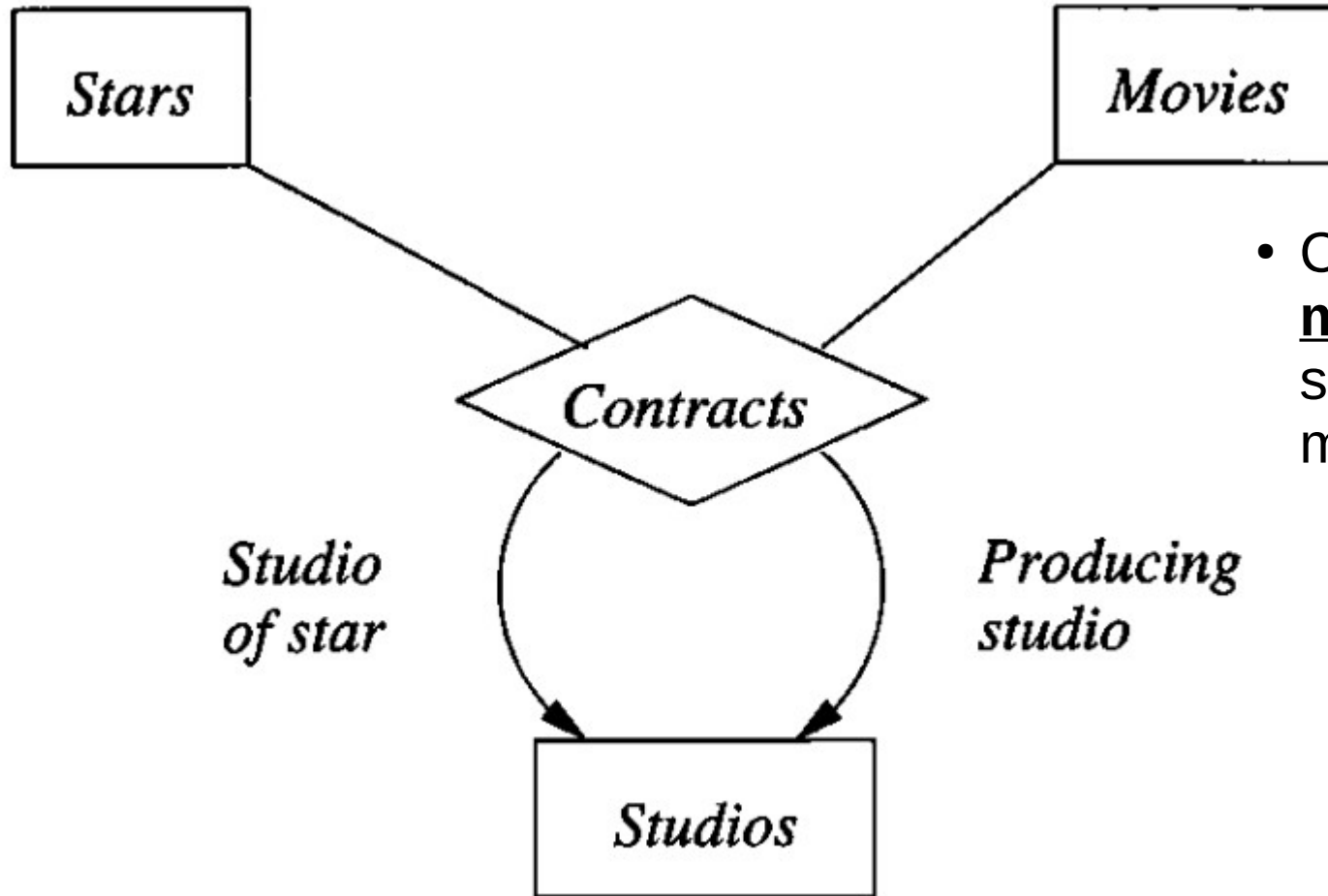
- Contracts (starName, movieTitle, movieYear, studioName)

# Relationship with Roles



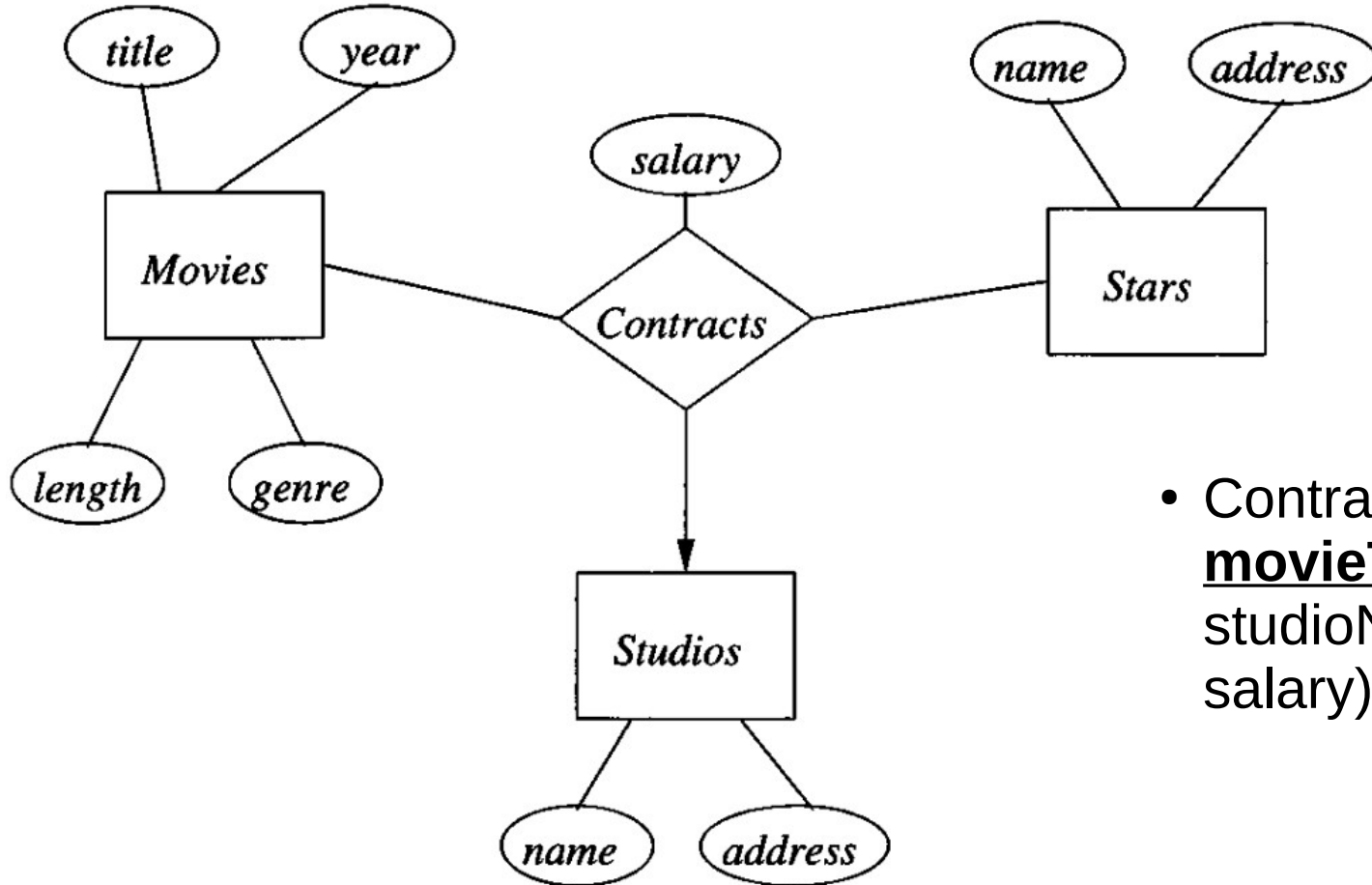
- Movies (title, year, length, genre, studioName, **originalTitle**, **originalYear**)

# Multi (Four)-Way Relationships



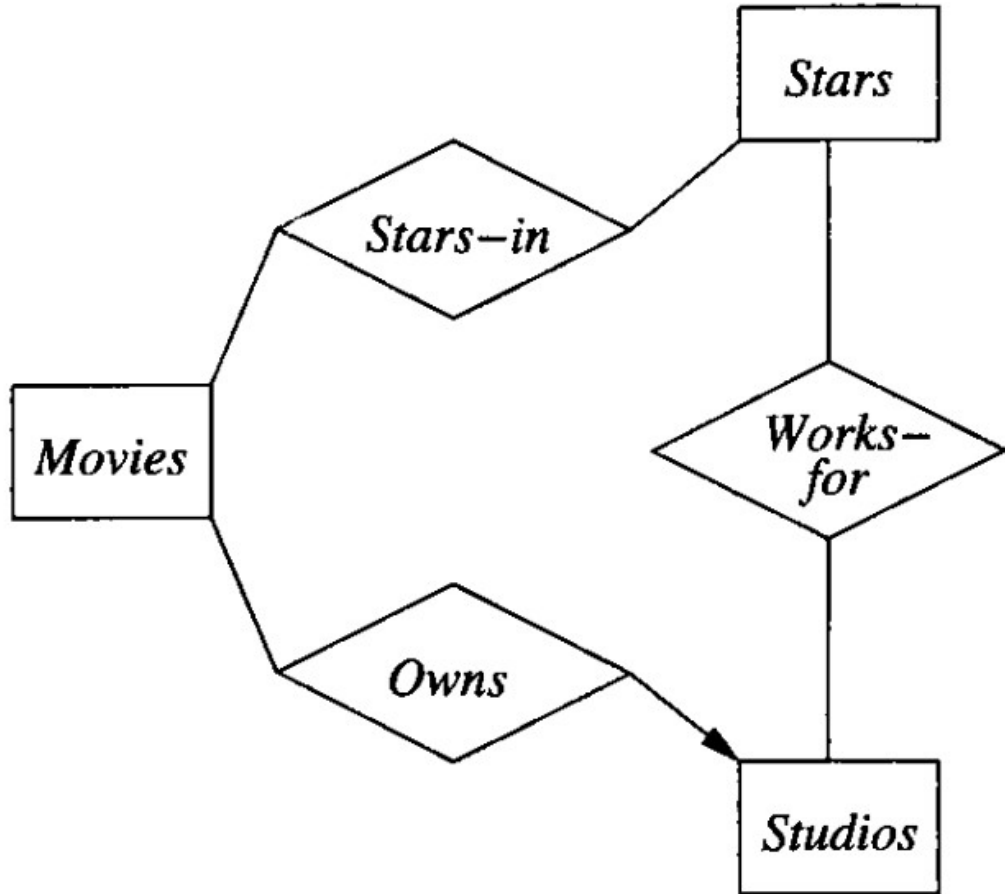
- Contracts (starName, movieTitle, movieYear, starStudioName, movieStudioName)

# Relationships with Attributes



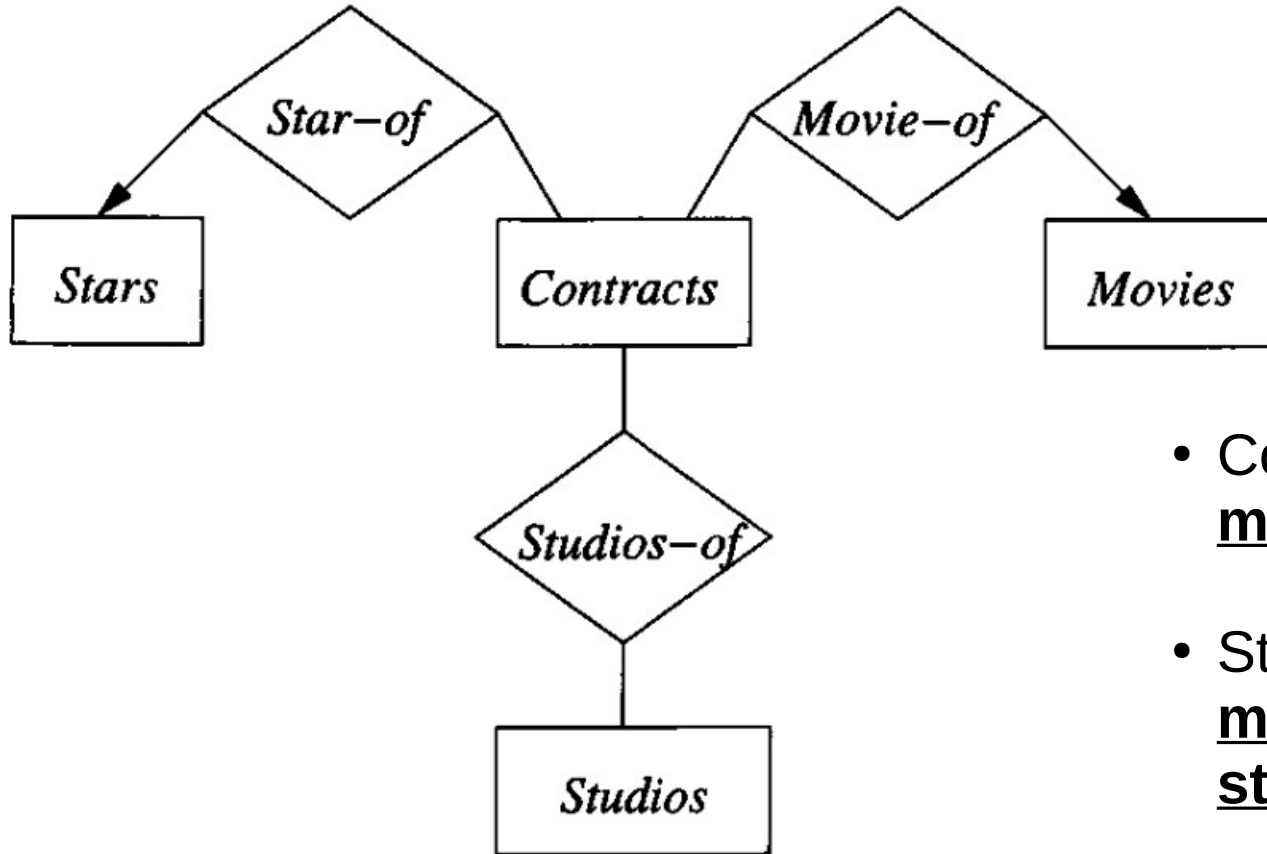
- Contracts (starName, movieTitle, movieYear, studioName, salary)

# E/R to Relations



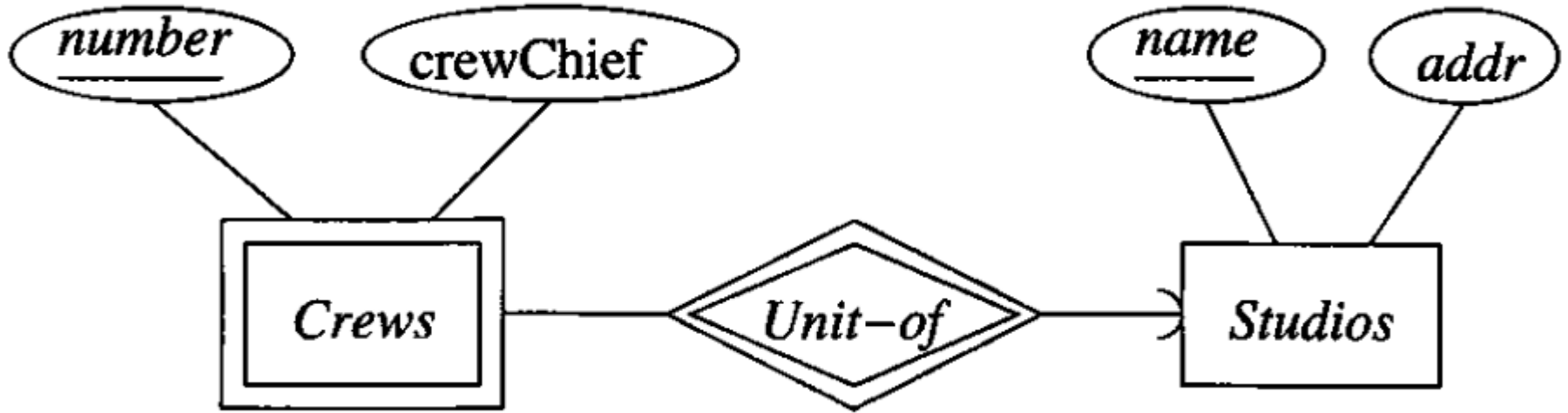
- Works-for (starName, studioName)

# Multi-Way Relationships



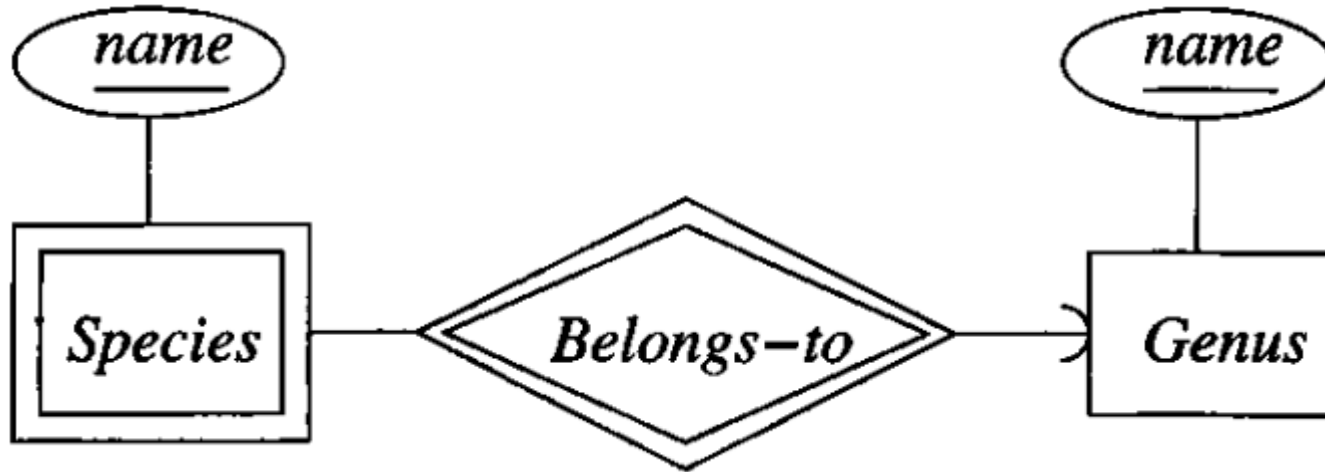
- Contracts (starName, movieTitle, movieYear)
- Studios-of (starName, movieTitle, movieYear, studioName)

# Weak Entities (1)



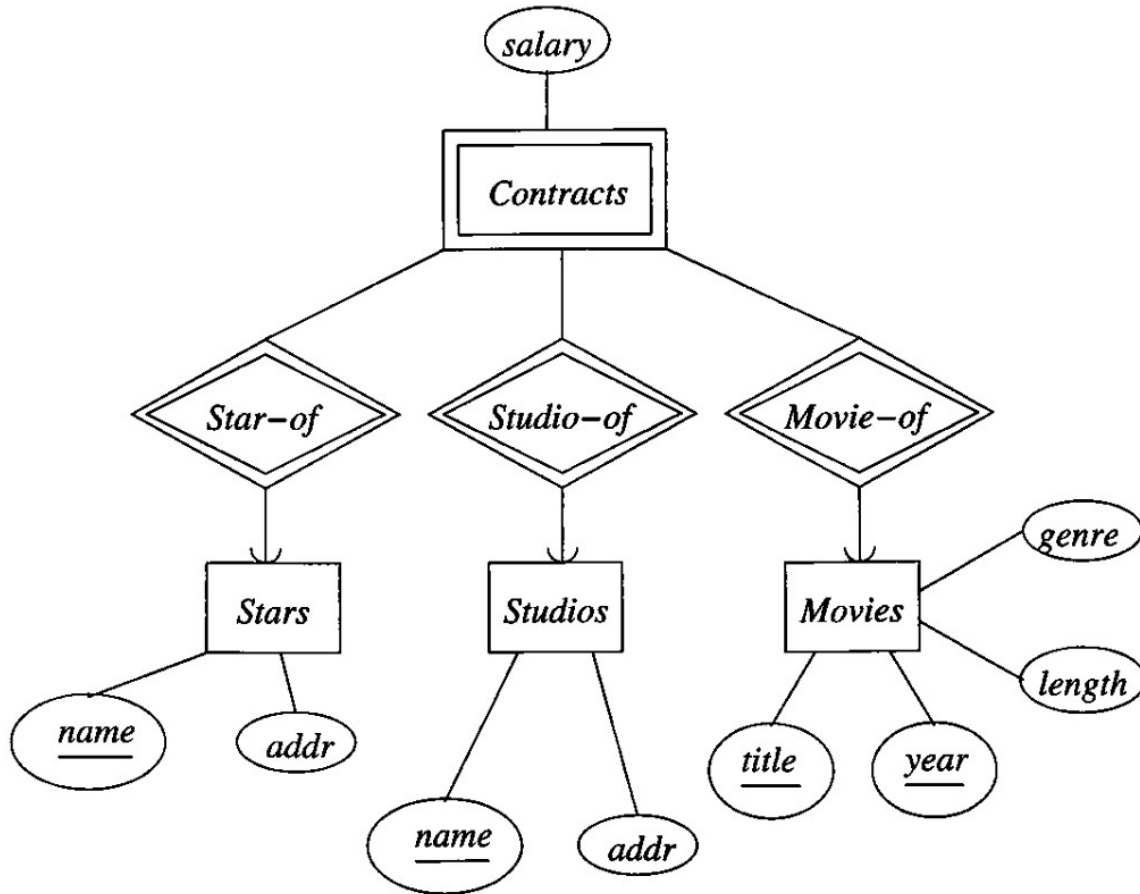
- Studios (**name**, *addr*)
- Crews (**studioName**, **number**, *crewChief*)

# Weak Entities (2)



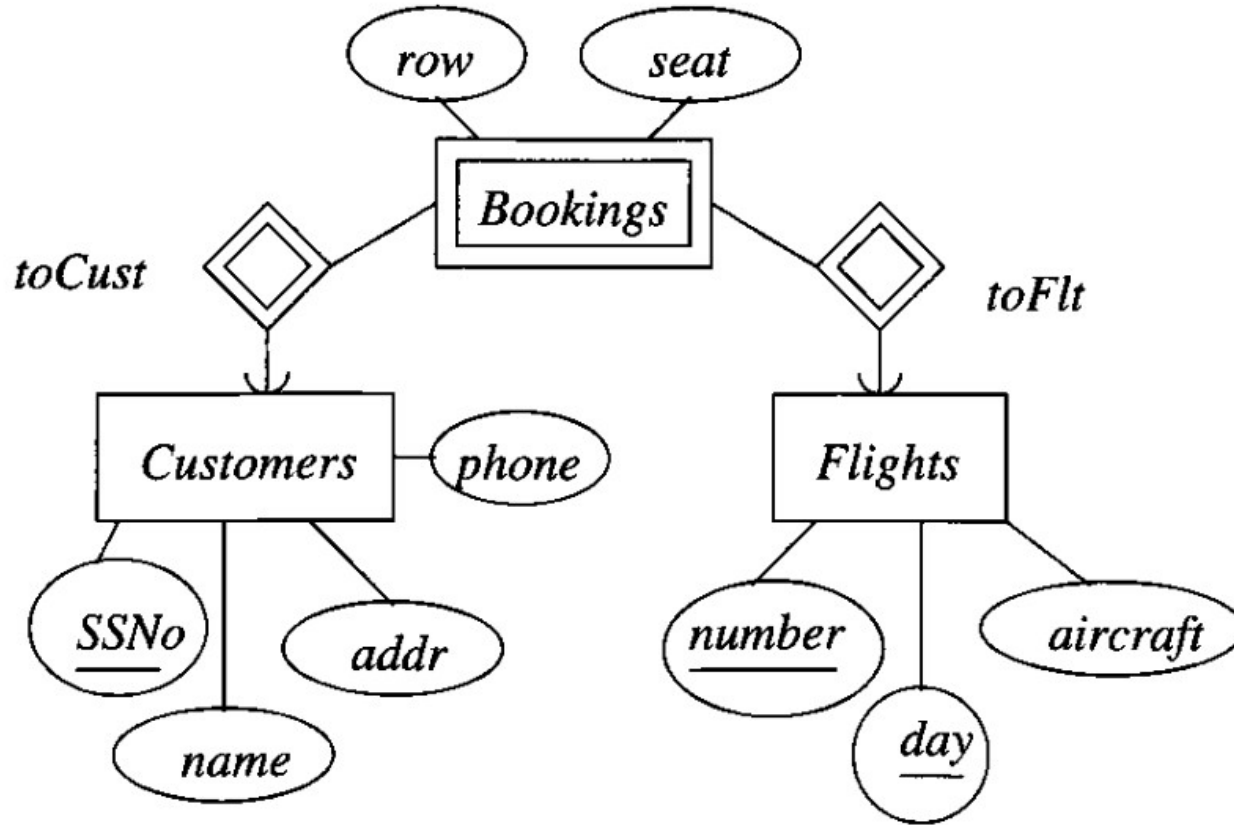
- Genus (name)
- Species (genusName, speciesName)

# Weak Entities (3)



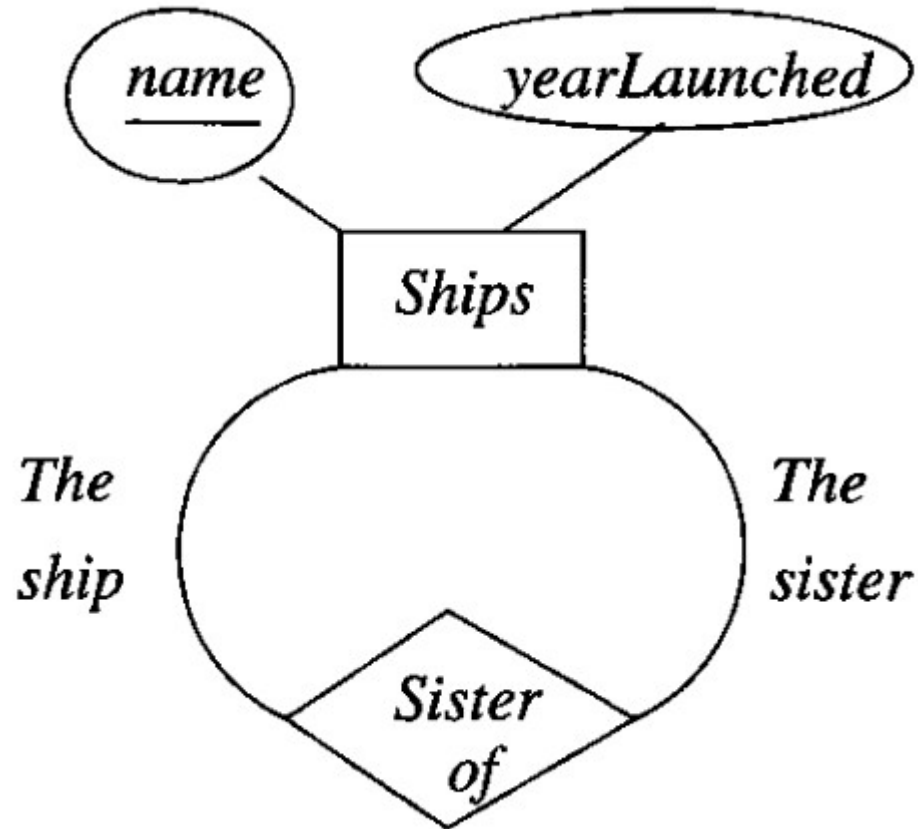
- Stars (name, addr)
- Studios (name, addr)
- Movies (title, year, genre, length)
- Contracts (starName, studioName, movieTitle, movieYear, salary)

# Example (1)



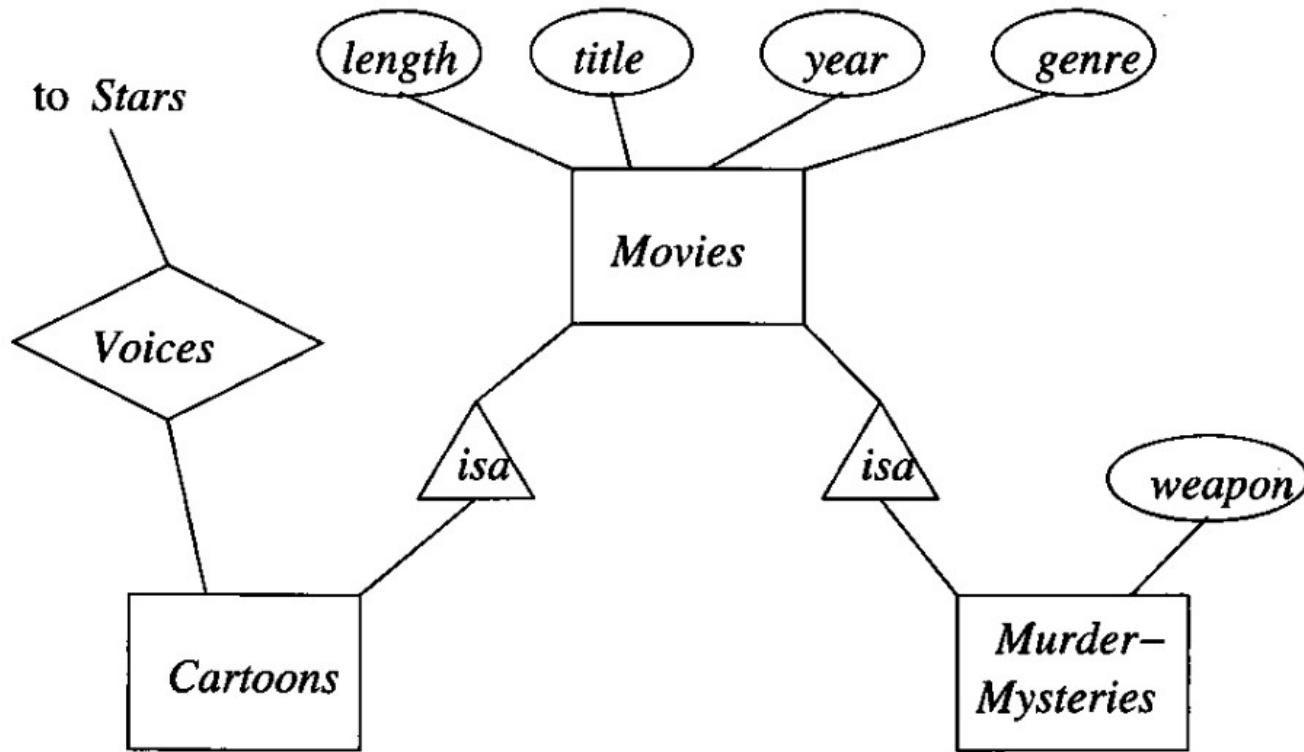
- Customers (SSNo, name, addr, phone)
- Flights (number, day, aircraft)
- Bookings (custSSNo, flightNo, flightDay, row, seat)

# Example (2)



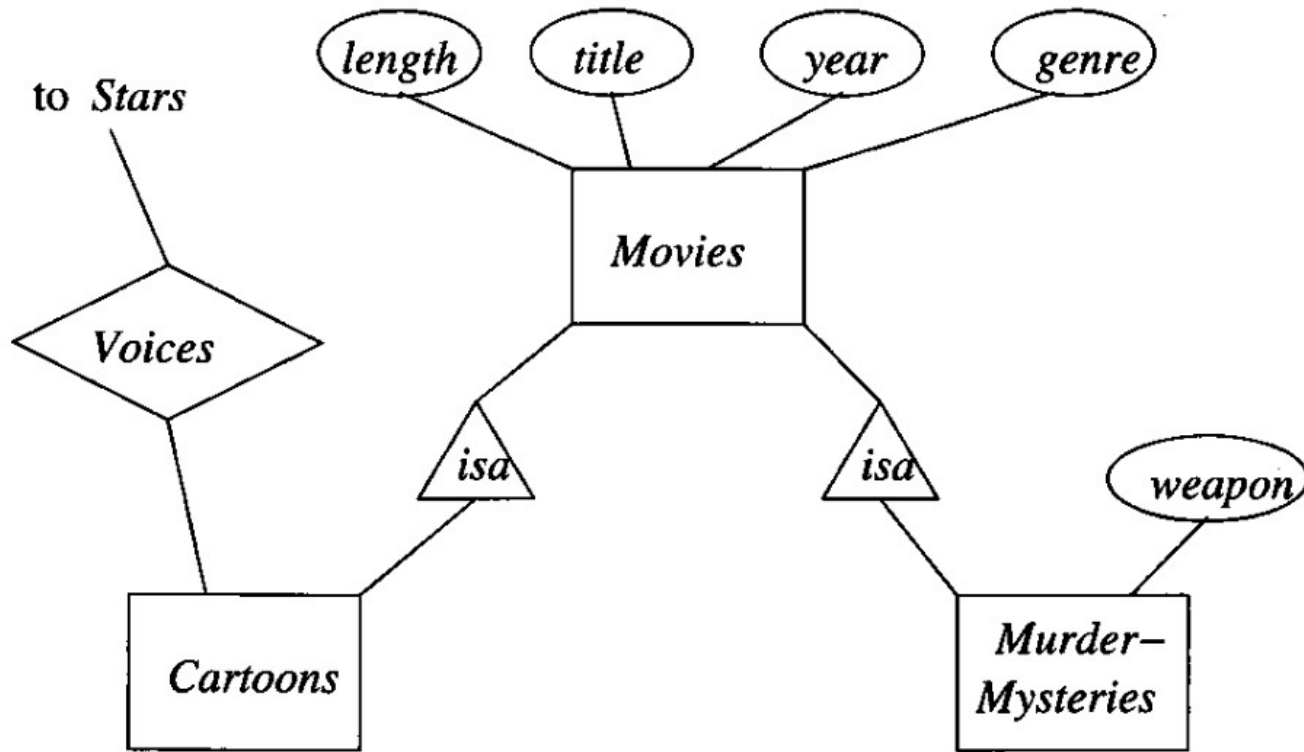
- Ships (**name**, yearLaunched)
- Sister-of (**shipName**, **sisterShipName**)

# ISA Relationships: E/R Style



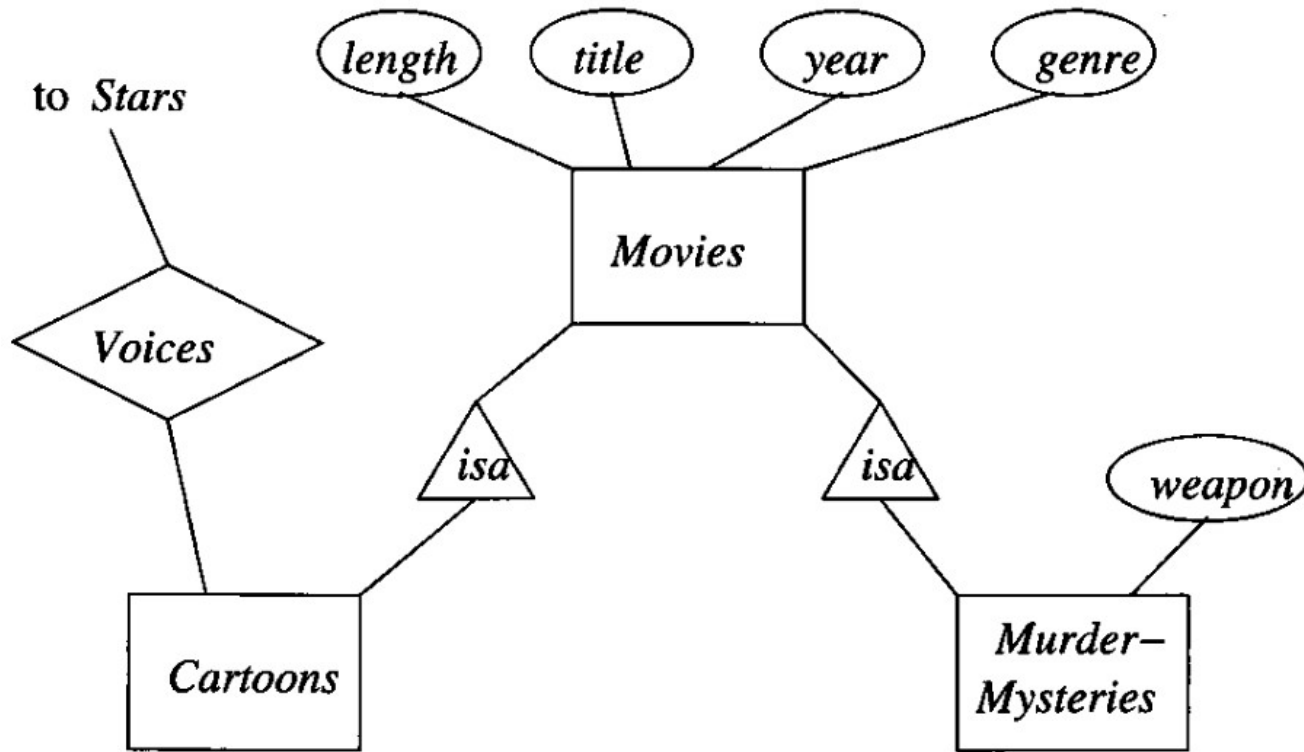
- Movies (title, year, genre, length)
- Cartoons (title, year)
- Murder-Mysteries (title, year, weapon)
- Stars (name, address)
- Voices (title, year, starName)

# ISA Relationships: Object-Oriented



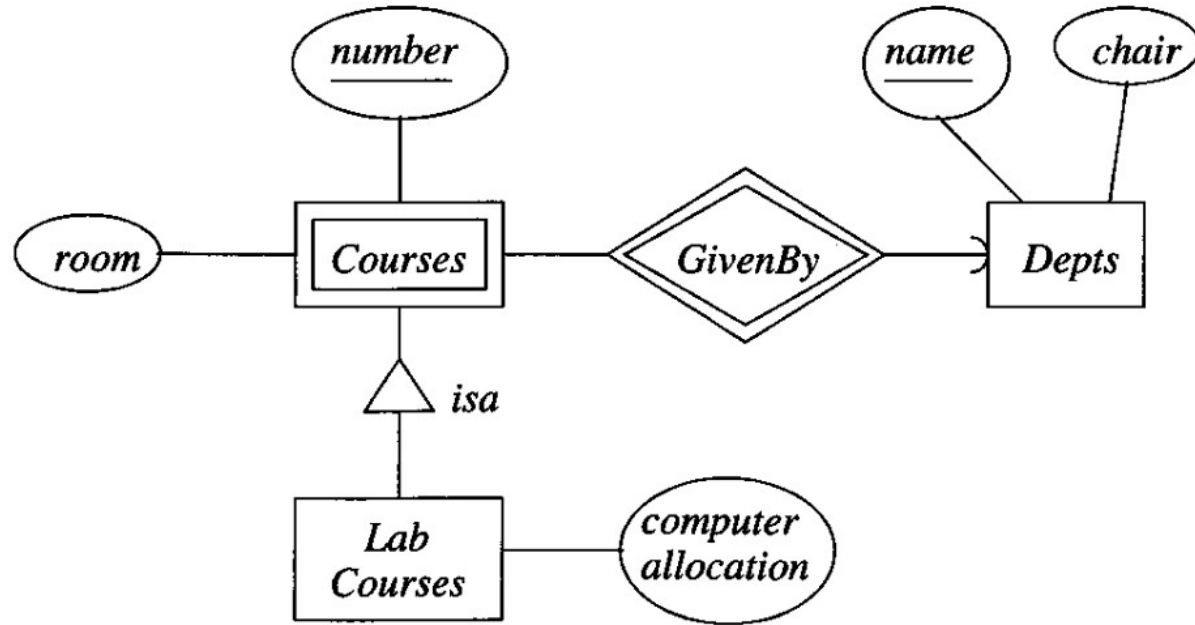
- Movies (**title**, **year**, genre, length)
- Cartoons (**title**, **year**, genre, length)
- Murder-Mysteries (**title**, **year**, genre, length, weapon)
- Cartoons-Murder-Mysteries (**title**, **year**, genre, length, weapon)

# ISA Relationships: NULLs



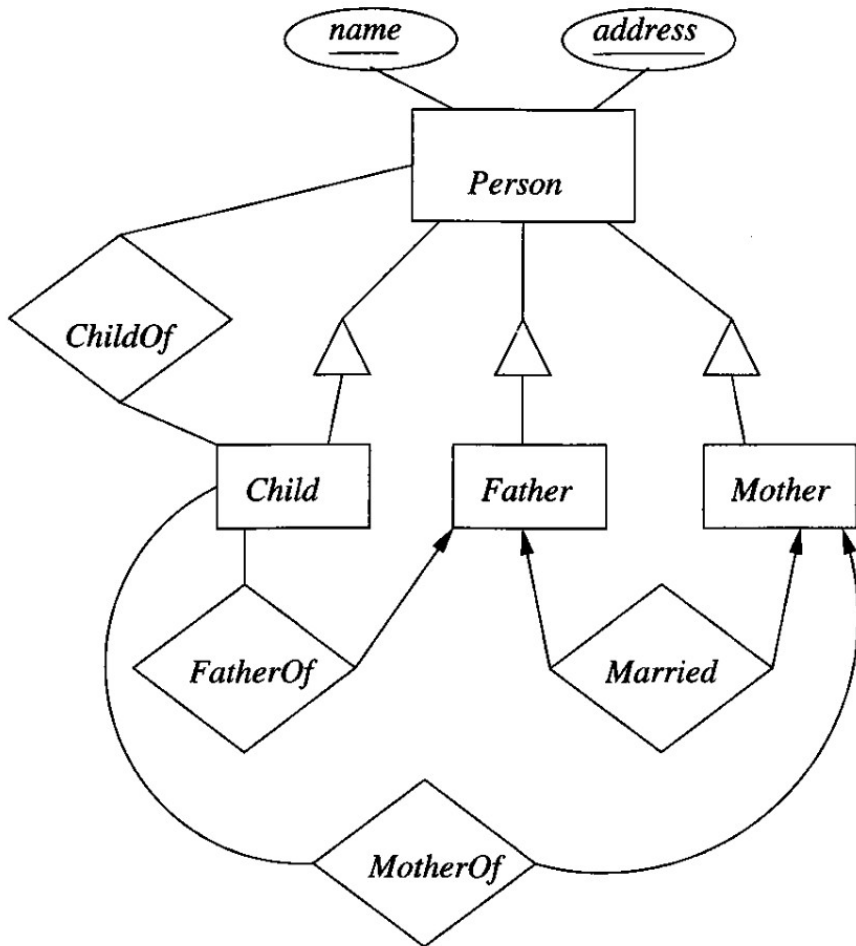
- *Movies* (*title*, *year*, *genre*, *length*, *weapon*)

# Example (3).



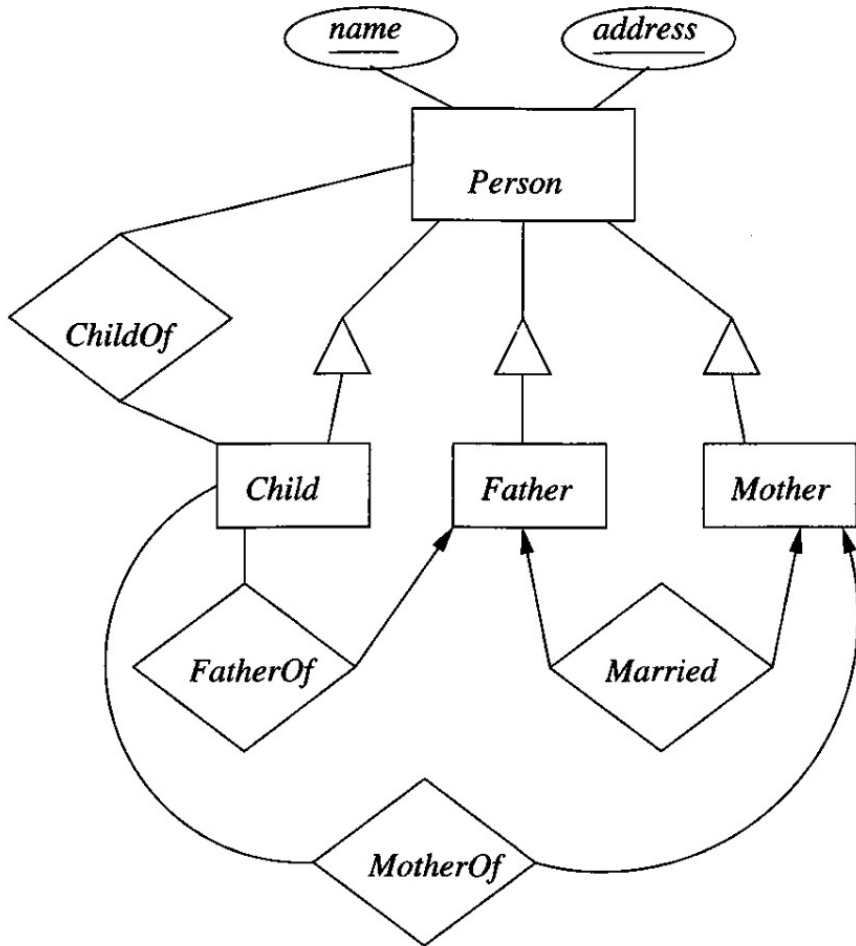
- Depts (name, chair)
- E/R style
  - Courses (deptName, number, room)
  - LabCourses (deptName, number, computerAllocation)
- Object-oriented
  - Courses (deptName, number, room)
  - LabCourses (deptName, number, room, computerAllocation)
- NULLs
  - Courses (deptName, number, room, computerAllocation)

# Example (4) E/R-style



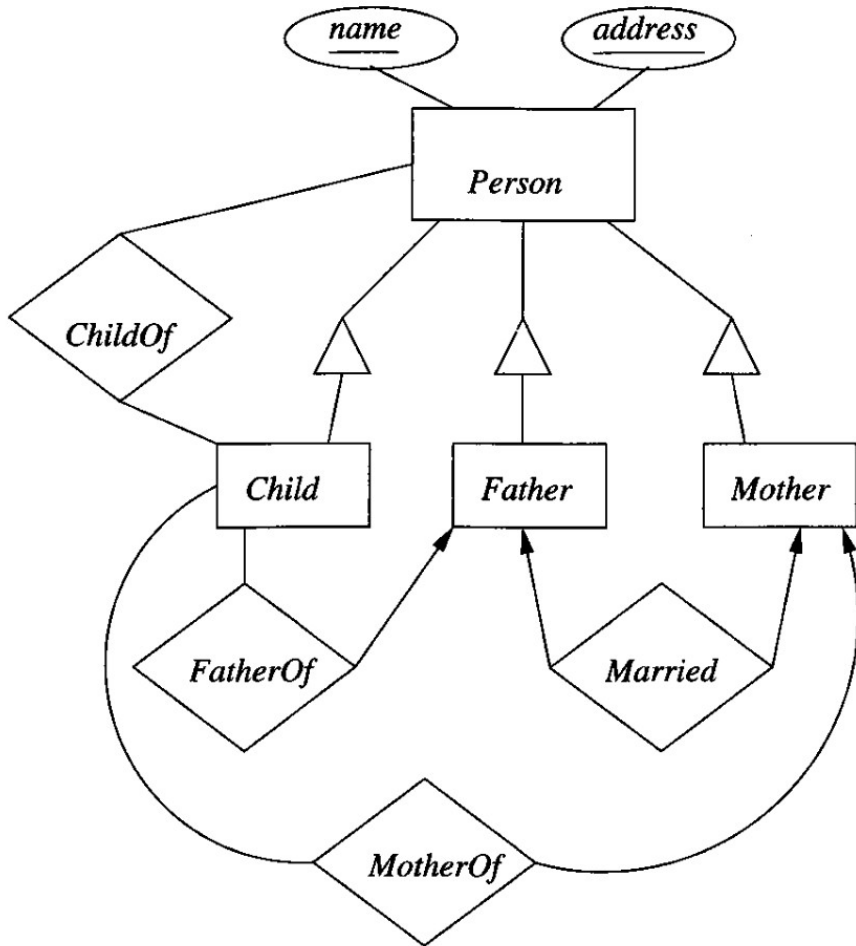
- Person (name, address)
- Child (name, address, fName, fAddr, mName, mAddr)
- Father (name, address, spouseName, spouseAddr)
- Mother (name, address)
- ChildOf (pName, pAddr, cName, cAddr)

# Example (4) Object-Oriented



- Person (name, address)
- Child (name, address, fName, fAddr, mName, mAddr)
- Father (name, address, spouseName, spouseAddr)
- Mother (name, address)
- ChildFather (name, address, fName, fAddr, mName, mAddr, spouseName, spouseAddr)
- ChildMother (name, address, fName, fAddr, mName, mAddr)

# Example (4) NULLs

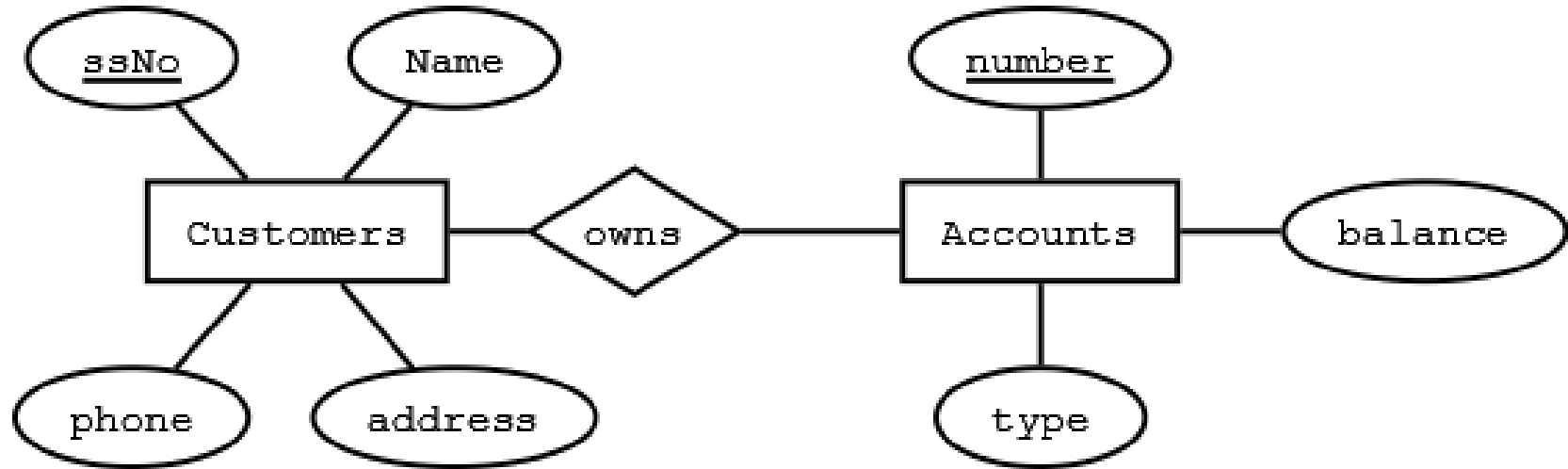


- Person (name, address, fName, fAddr, mName, mAddr, spouseFName, spouseFAddr, spouseMName, spouseMAddr)

# E/R Diagrams

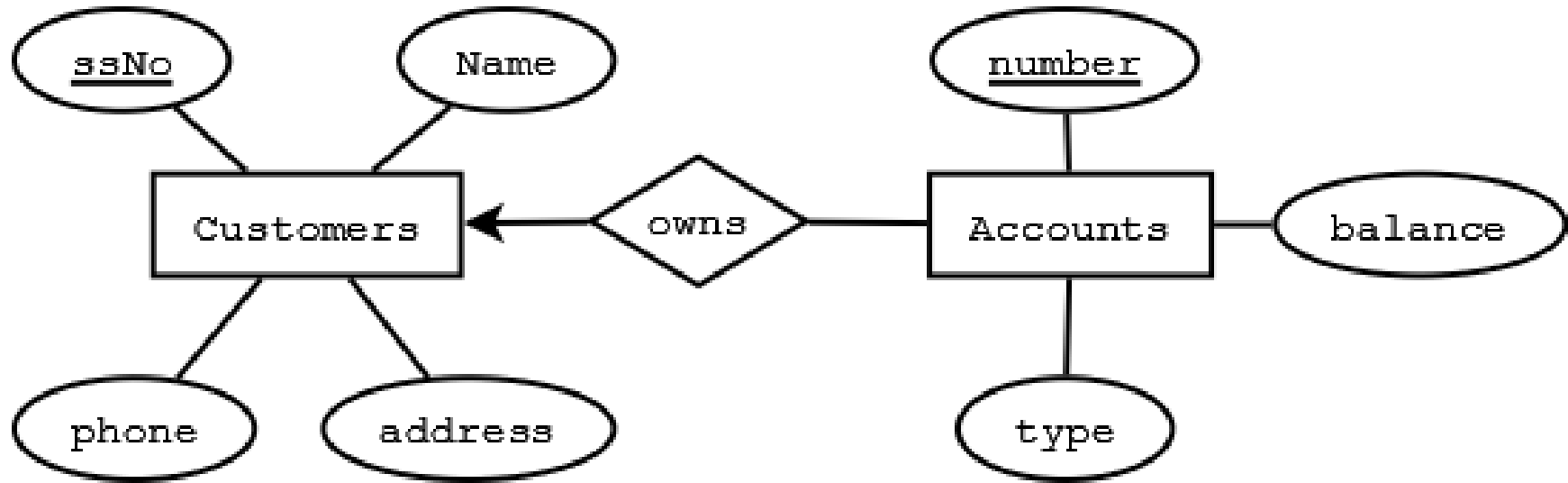
## Examples

# Exercise 4.1.1



- Customers (ssNo, name, phone, address)
- Accounts (number, type, balance)
- Owns (ssNo, acctNo)

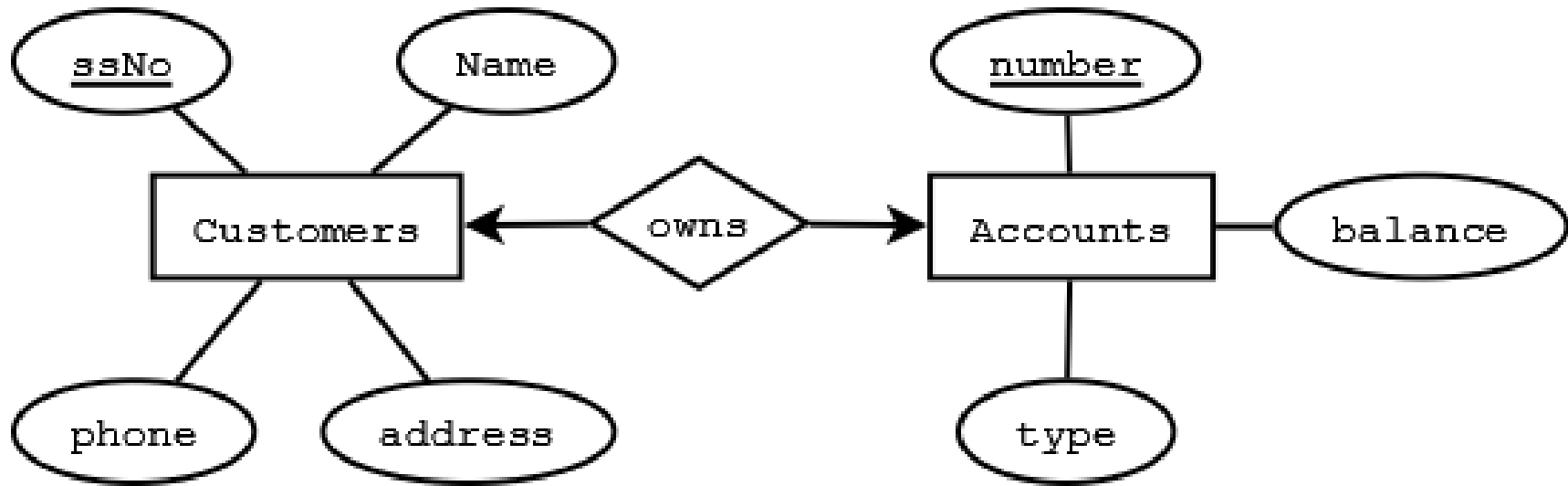
# Exercise 4.1.2 a



- Accounts (number, type, balance, **ssNo**)
- Customers (ssNo, name, phone, address)

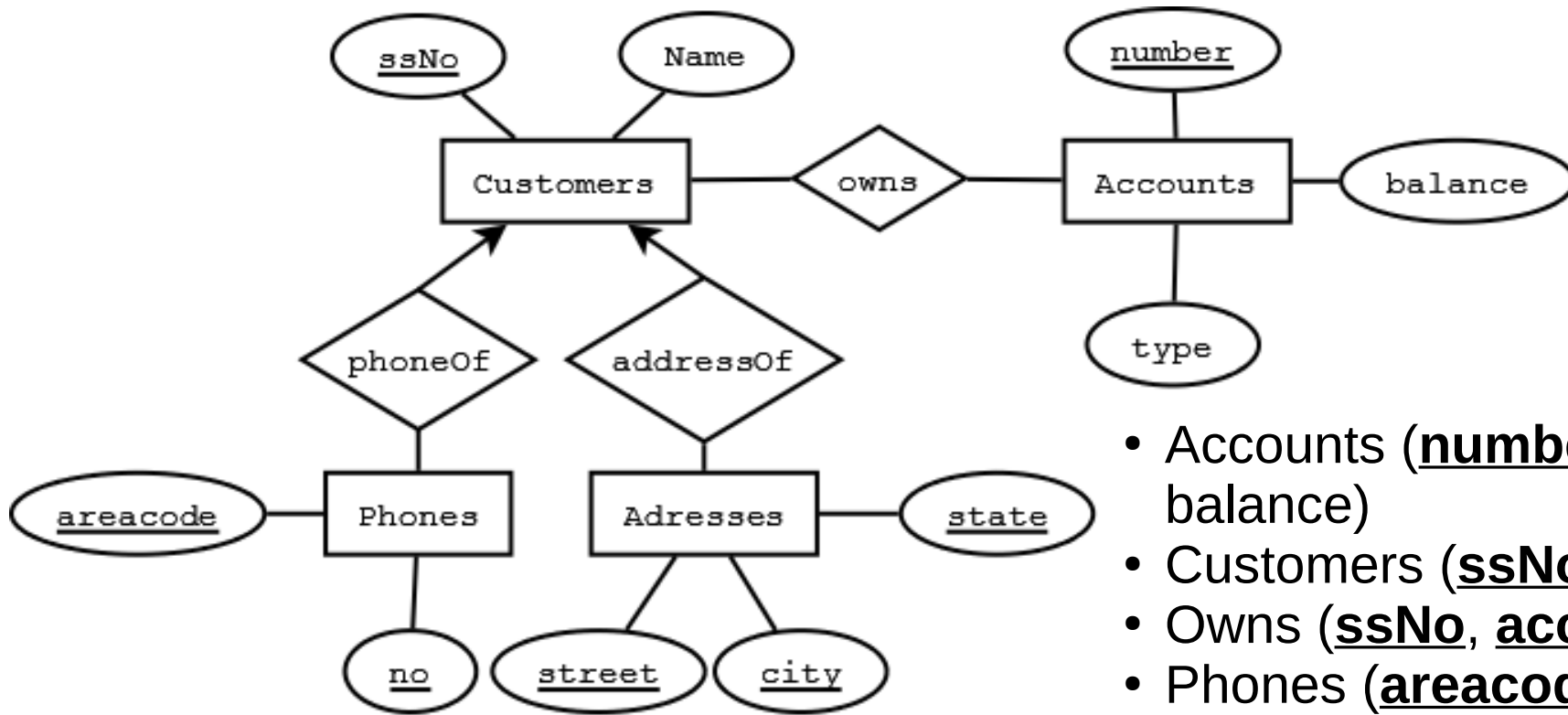
# Exercise 4.1.2 b

- Accounts (number, type, balance, **ssNo**)



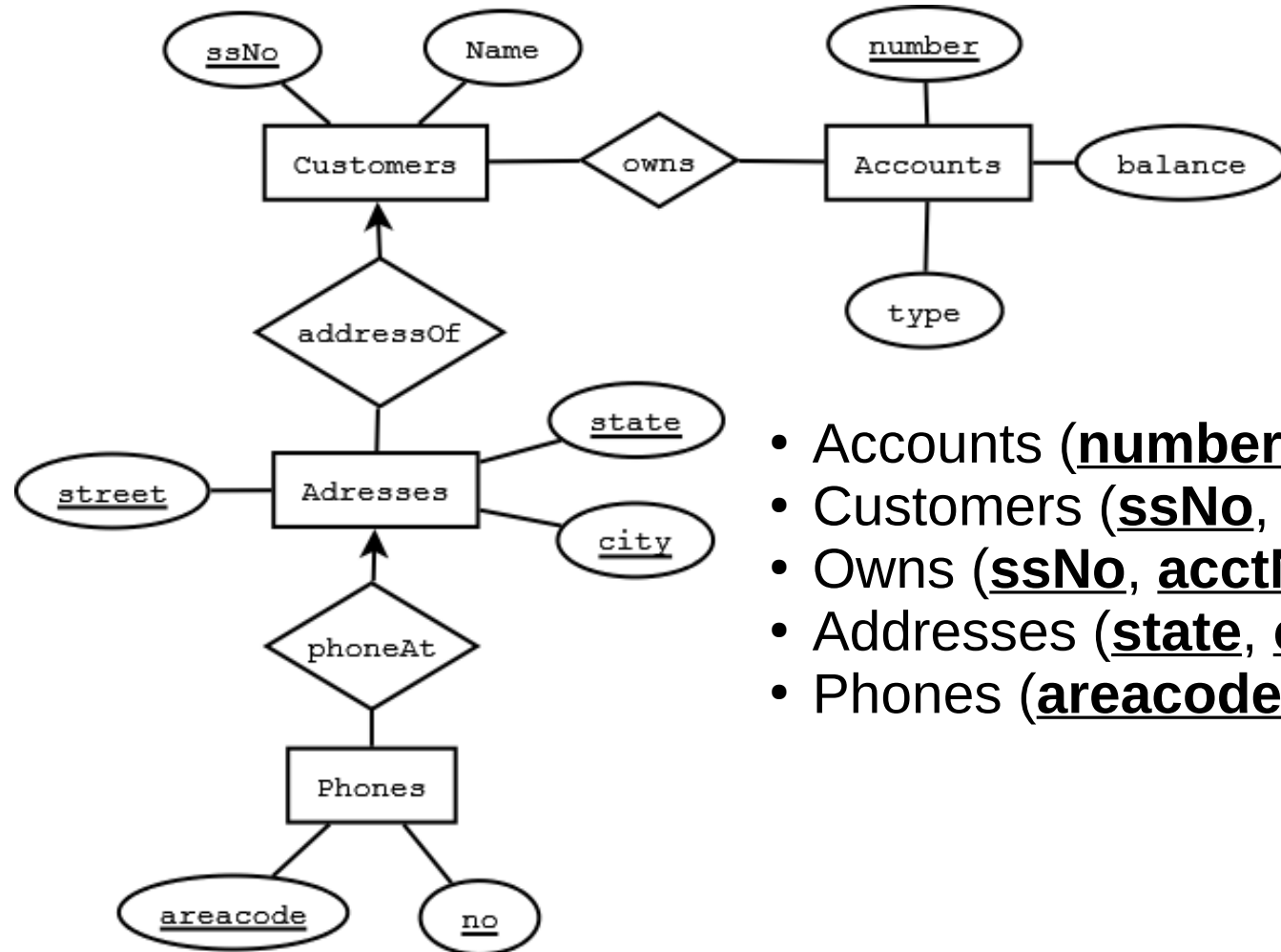
- Customers (**ssNo**, name, phone, address, **acctNumber**)

# Exercise 4.1.2 c



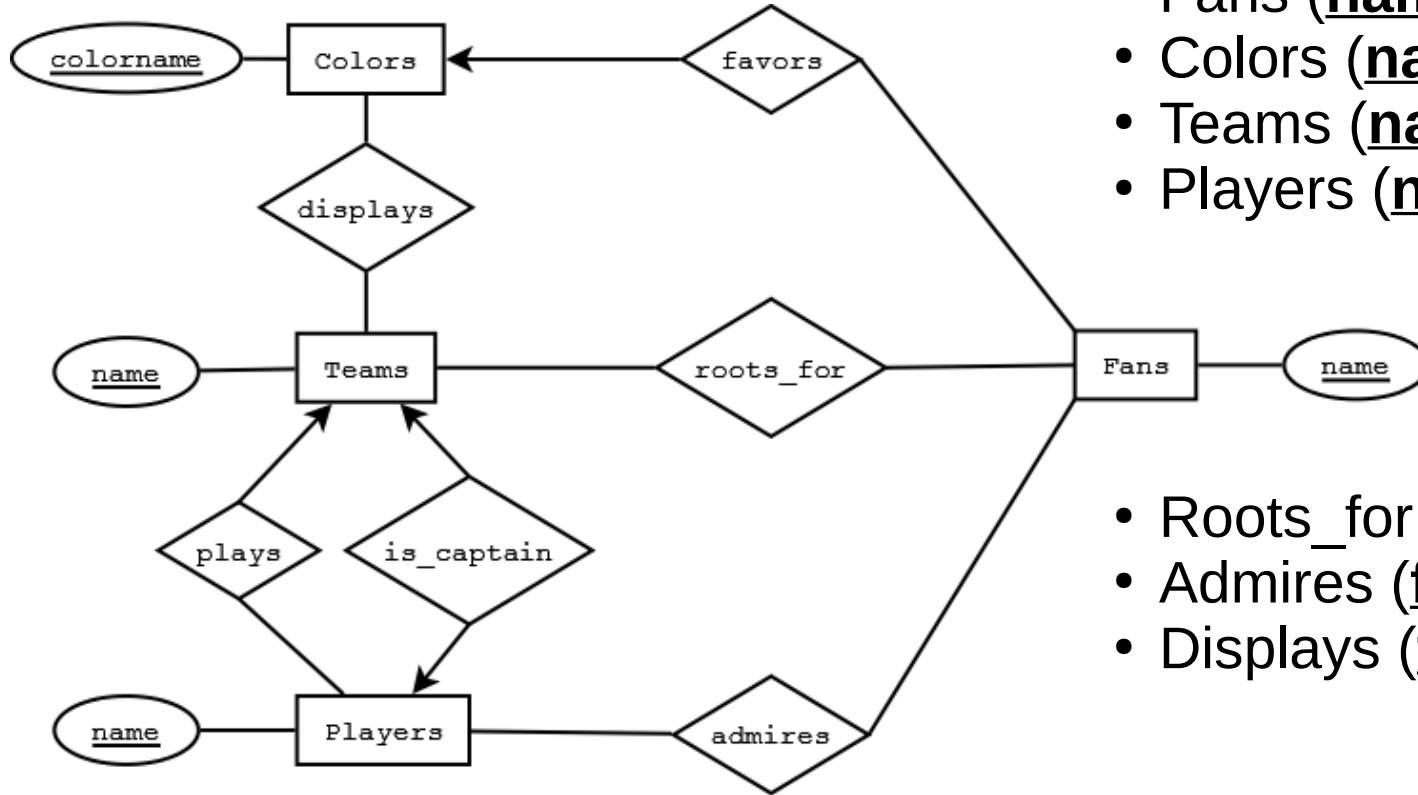
- Accounts (number, type, balance)
- Customers (ssNo, name)
- Owns (ssNo, acctNo)
- Phones (areacode, no, **ssNo**)
- Addresses (state, city, street, **ssNo**)

# Exercise 4.1.2 d



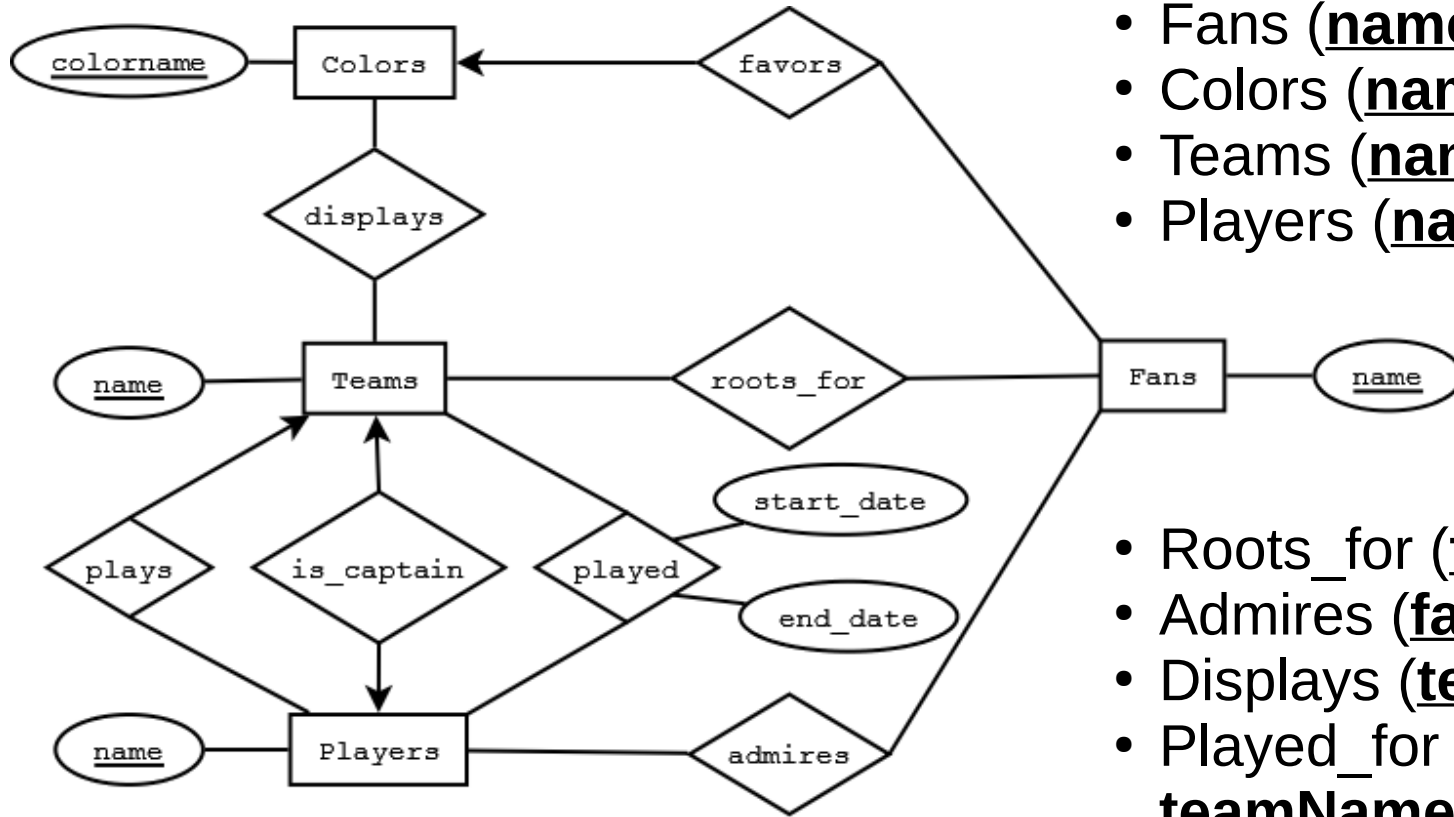
- Accounts (number, type, balance)
- Customers (ssNo, name)
- Owns (ssNo, acctNo)
- Addresses (state, city, street, **ssNo**)
- Phones (areacode, no, **state**, **city**, **street**)

# Exercise 4.1.3



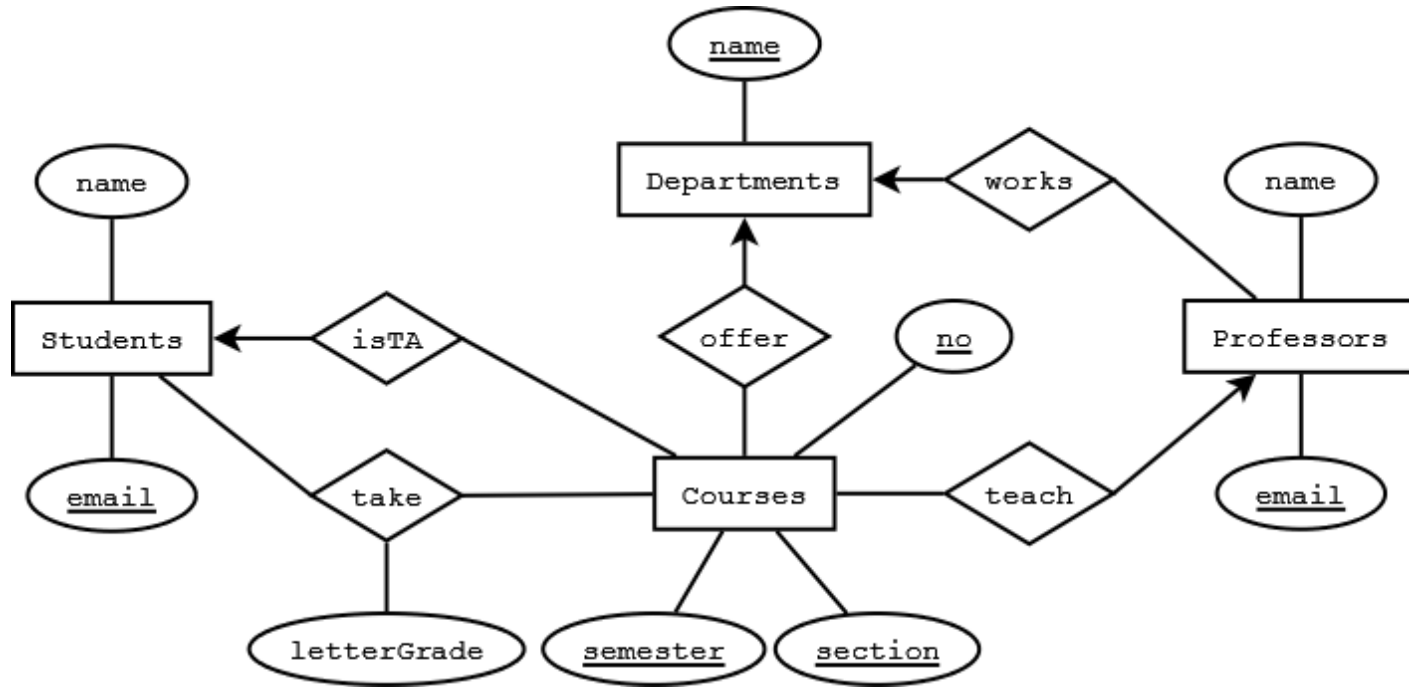
- Fans (name, favoriteColor)
  - Colors (name)
  - Teams (name, captainPName)
  - Players (name, teamName)
- 
- Roots\_for (fanName, teamName)
  - Admires (fanName, playerName)
  - Displays (teamName, colorName)

# Exercise 4.1.5



- Fans (name, favoriteColor)
  - Colors (name)
  - Teams (name, captainPName)
  - 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 (email, name)
- Courses (no, semester, section, TAemail, deptName, profEmail)
- Departments (name)
- Professors (email, name, deptName)

- Take (studentEmail, cNo, cSemester, cSection, 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)  $\rightarrow$  Output schema
  - Input tuples  $\rightarrow$  Output tuples
- Single table operations
  - Select column, select tuple (row), aggregate, grouping
- Multiple table operations
  - Product and Join, Union, Intersection, Difference

# Projection $\pi$

- Input table

- $T(A,B,C)$

- **A B C**

1 2 3

3 4 6

8 5 4

7 4 3

- $T' = \pi_{A, (A+B+C) \text{ AS } S'}(T)$

- Output table:  $T'$

- Schema

- $T'(A,S')$

- Same number of tuples as  $T$

- No duplicate elimination

- **A S'**

1 6

3 13

8 17

7 14

# Selection $\sigma$

- 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 $\delta$

- Input table
- $T' = \delta(T)$

T(A,B)

0 1

2 3

0 1

2 4

3 4

- 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)

0 1

2 3

2 4

3 4

# Sorting $\tau$

- 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

- $T' = \text{SUM}_A(T)$

- $T'' = \text{MAX}_{A+B}(T)$

- Output table: T'

- Schema

- T'(X)

- Single tuple with aggregate result

T'(X)

7

T''(X)

7

# GroupBy Aggregations $\gamma$

- Input table

T(A,B)

0 1

2 3

0 1

2 4

3 4

- $T' = \gamma_{A, \text{MIN}(B) \text{ AS } MB}(T)$

- Output table: T'

- Schema

- T'(A, MB)

- Arguments of  $\gamma$

- Tuples have distinct values for A and group aggregate value for other attributes

T'(A,MB)

0 1

2 3

3 4

# Set Operations $\cup$ , $\cap$ , $-$

- 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:  $T' = R \cup S$

1 1  
1 2  
3 4  
4 3

- Difference:  $T' = R - S$

1 1  
3 4  
4 3

- Difference:  $T' = S - R$

- Intersection:  $T' = R \cap S$

1 2

# Cartesian Product $\times$

- $R(A) = \{1,1,2,3\}$
- $S(B) = \{1,3,4\}$
- $T = R \times S(A,B) = \{$   
     $(1,1),(1,3),(1,4),$   
     $(1,1),(1,3),(1,4),$   
     $(2,1),(2,3),(2,4),$   
     $(3,1),(3,3),(3,4)\}$
- 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  $R \times S$  is  $|R| \times |S|$ , 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
  - $R(A)$
  - $S(B)$
  - $T(A,B) = A \cup B$

# Join $\bowtie$

- $R(A) = \{1,1,2,3\}$
- $S(B) = \{1,3,4\}$
- $T = R \bowtie_{A=B} S = \{$   
 $(\textcolor{red}{1},\textcolor{green}{1}),(\textcolor{red}{1},\textcolor{green}{3}),(\textcolor{red}{1},\textcolor{green}{4}),$   
 $(\textcolor{red}{1},\textcolor{green}{1}),(\textcolor{red}{1},\textcolor{green}{3}),(\textcolor{red}{1},\textcolor{green}{4}),$   
 $(\textcolor{red}{2},\textcolor{green}{1}),(\textcolor{red}{2},\textcolor{green}{3}),(\textcolor{red}{2},\textcolor{green}{4}),$   
 $(\textcolor{red}{3},\textcolor{green}{1}),(\textcolor{red}{3},\textcolor{green}{3}),(\textcolor{red}{3},\textcolor{green}{4})\} = \{(\textcolor{red}{1},\textcolor{green}{1}),(\textcolor{red}{1},\textcolor{green}{1}),(\textcolor{red}{3},\textcolor{green}{3})\}$
- 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
  - $R(A)$
  - $S(B)$
  - $T(A,B) = A \cup B$
- $R \bowtie_{A=B} S = \sigma_{A=B}(R \times S)$

# Outer Joins

| $R(A,B)$ | $S(B,C)$ | $R \bowtie S$<br>[natural join]<br>(A,B,C) | $R \bowtie_o S$ [full outer<br>join] (A,B,C) |
|----------|----------|--------------------------------------------|----------------------------------------------|
| 0 1      | 0 1      |                                            | 2 3 4                                        |
| 2 3      | 2 4      |                                            | 2 3 4                                        |
| 0 1      | 2 5      | 2 3 4                                      | 0 1 -                                        |
| 2 4      | 3 4      | 2 3 4                                      | 0 1 -                                        |
| 3 4      | 0 2      |                                            | 2 4 -                                        |
|          | 3 4      |                                            | 3 4 -                                        |
|          |          |                                            | - 0 1                                        |
|          |          |                                            | - 2 4                                        |
|          |          |                                            | - 2 5                                        |
|          |          |                                            | - 0 2                                        |

# Left (Right) Outer Joins

$R \bowtie_o S$  [full outer join]

$R(A,B)$     $S(B,C)$

| $R(A,B)$ | $S(B,C)$ | $(A,B,C)$ |
|----------|----------|-----------|
| 0 1      | 0 1      | 2 3 4     |
| 2 3      | 2 4      | 2 3 4     |
| 0 1      | 2 5      | 0 1 -     |
| 2 4      | 3 4      | 2 4 -     |
| 3 4      | 0 2      | 3 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 - |
| 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 $\leftrightarrow$ SQL

- SELECT  $\leftrightarrow$  Projection  $\pi$
- FROM  $\leftrightarrow$  Input tables
- WHERE  $\leftrightarrow$  Selection  $\sigma$ , Join predicates
- DISTINCT  $\leftrightarrow$  Duplicate elimination  $\delta$
- ORDER BY  $\leftrightarrow$  Sorting  $\tau$
- GROUP BY  $\leftrightarrow$  GroupBy aggregations  $\gamma$
- UNION, INTERSECT, EXCEPT  $\leftrightarrow$  Set operations  $\cup, \cap, -$
- JOIN  $\leftrightarrow$  Join

Relational Algebra  
Expressions = Queries

# Relational Algebra Operators

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\cup, \cap, -$
- Product  $\times$
- Join  $\bowtie$
- 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  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\mathbf{U}$ ,  $\cap$ ,  $-$
- Product  $\mathbf{x}$
- Join  $\bowtie$

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

## 2.4.1 b)

- Projection  $\pi$
  - Selection  $\sigma$
  - Duplicate elimination  $\delta$
  - Sorting  $\tau$
  - GroupBy aggregations  $\gamma$
  - Set operations  $\mathbf{U}$ ,  $\cap$ ,  $-$
  - Product  $\mathbf{x}$
  - Join  $\bowtie$
- $S_1(M, S, R, H, Sc, P) = \sigma_{H \geq 100}(\text{Laptop}(M, S, R, H, Sc, P))$   
 $S_2(Ma, M, T, S, R, H, Sc, P) = \text{Product}(Ma, M, T) \bowtie S_1(M, S, R, H, Sc, P)$   
 $R(\text{maker}) = \pi_{Ma}(S_2(Ma, M, T, S, R, H, Sc, P))$
  - $R(\text{maker}) = \pi_{\text{maker}}(\text{Product} \bowtie \sigma_{hd \geq 100}(\text{Laptop}))$

## 2.4.1 c)

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\cup, \cap, -$
- Product  $\times$
- Join  $\bowtie$

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

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

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

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

## 2.4.1 d)

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\cup, \cap, -$
- Product  $\times$
- Join  $\bowtie$

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

$$R(\text{model}) = \pi_M(S_1(M, C, T, P))$$

- $R(\text{model}) = \pi_{\text{model}}(\sigma_{\text{color}=\text{true AND type}=\text{'laser'}}(\text{Printer}))$

## 2.4.1 e)

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\cup, \cap, -$
- Product  $\times$
- Join  $\bowtie$

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

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

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

## 2.4.1 f)

- Projection  $\pi$
  - Selection  $\sigma$
  - Duplicate elimination  $\delta$
  - Sorting  $\tau$
  - GroupBy aggregations  $\gamma$
  - Set operations  $\mathbf{U}, \cap, -$
  - Product  $\mathbf{x}$
  - Join  $\bowtie$
- $S_1(\text{hd}, \text{cnt}) = \gamma_{\text{hd}, \text{COUNT}(\ast) \text{ AS cnt}}(\text{PC})$   
 $S_2(\text{hd}, \text{cnt}) = \sigma_{\text{cnt} \geq 2}(S_1)$   
 $R(\text{hd}) = \pi_{\text{hd}}(S_2)$
  - $R(\text{hd}) = \pi_{\text{hd}}(\sigma_{\text{cnt} \geq 2}(\gamma_{\text{hd}, \text{COUNT}(\ast) \text{ AS cnt}}(\text{PC})))$

## 2.4.1 g)

- Projection  $\pi$
  - Selection  $\sigma$
  - Duplicate elimination  $\delta$
  - Sorting  $\tau$
  - GroupBy aggregations  $\gamma$
  - Set operations  $\mathbf{U}, \cap, -$
  - Product  $\mathbf{x}$
  - Join  $\bowtie$
- $S_1(M_1, Sp_1, R_1, H_1, P_1, M_2, Sp_2, R_2, H_2, P_2) =$   
 $PC \rightarrow PC_1(M_1, Sp_1, R_1, H_1, P_1)$   
 $\bowtie_{Sp1=Sp2 \text{ AND } R1=R2 \text{ AND } M1 < M2}$   
 $PC \rightarrow PC_2(M_2, Sp_2, R_2, H_2, P_2)$   
 $R(model_1, model_2) =$   
 $\pi_{M1, M2}(S_1(M_1, Sp_1, R_1, H_1, P_1,$   
 $M_2, Sp_2, R_2, H_2, P_2))$

## 2.4.1 h)

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\cup, \cap, -$
- Product  $\times$
- Join  $\bowtie$

- $S_1(\text{model}, \text{maker}) = \pi_{\text{model}, \text{maker}}(\text{Product} \bowtie \sigma_{\text{speed} \geq 2.8}(\text{PC}))$
- $S_2(\text{model}, \text{maker}) = \pi_{\text{model}, \text{maker}}(\text{Product} \bowtie \sigma_{\text{speed} \geq 2.8}(\text{Laptop}))$
- $S_3(\text{model}, \text{maker}) = S_1 \cup S_2$
- $S_4(\text{maker}, \text{cnt}) = \gamma_{\text{maker}, \text{COUNT}(*)} \text{AS cnt}(S_3)$
- $S_5(\text{maker}, \text{cnt}) = \sigma_{\text{cnt} \geq 2}(S_4)$
- $R(\text{maker}) = \pi_{\text{maker}}(S_5)$

## 2.4.1 i)

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\mathbf{U}, \cap, -$
- Product  $\mathbf{x}$
- Join  $\bowtie$

- $S_1(\text{model}, \text{speed}) = \pi_{\text{model}, \text{speed}}(\text{PC})$   
 $S_2(\text{model}, \text{speed}) = \pi_{\text{model}, \text{speed}}(\text{Laptop})$   
 $S_3(\text{model}, \text{speed}) = S_1 \cup S_2$   
 $S_4(M_1, Sp_1, M_2, Sp_2) = S_3 \rightarrow S_{31}(M_1, Sp_1)$   
 $\bowtie_{Sp_1 < Sp_2 \text{ AND } M_1 < M_2} S_3 \rightarrow S_{32}(M_2, Sp_2)$   
 $S_5(\text{model}) = \pi_{M_1}(S_4(M_1, Sp_1, M_2, Sp_2))$   
 $S_6(\text{model}) = \pi_{\text{model}}(S_1) \cup \pi_{\text{model}}(S_2)$   
 $S_7 = S_6 - S_5$   
 $R(\text{maker}) = \pi_{\text{maker}}(\text{Product} \bowtie S_7)$

## 2.4.1 j)

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\mathbf{U}$ ,  $\cap$ ,  $-$
- Product  $\mathbf{x}$
- Join  $\bowtie$

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

$$S_2 = \delta(S_1)$$

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

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

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

## 2.4.1 k)

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\mathbf{U}, \cap, -$
- Product  $\mathbf{x}$
- Join  $\bowtie$

- $S_1(\text{maker,model}) = \pi_{\text{maker,model}}(\sigma_{\text{type}='pc'}(\text{Product}))$   
 $S_2(\text{maker,cnt}) = \gamma_{\text{maker, COUNT(*) AS cnt}}(S_1)$   
 $S_3(\text{maker,cnt}) = \sigma_{\text{cnt}=3}(S_2)$   
 $R(\text{maker}) = \pi_{\text{maker}}(S_3)$

# Relational Algebra Query Execution Trees

# Relational Algebra Operators

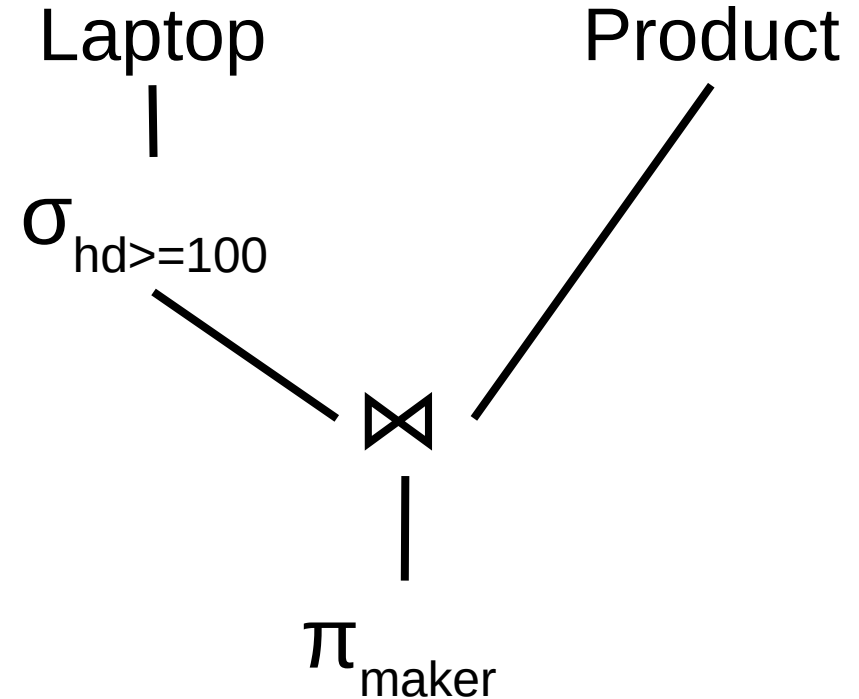
- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\cup, \cap, -$
- Product  $\times$
- Join  $\bowtie$
- Every operator takes as input one or two tables and generates as output a table
  - Schema
  - Tuples
- 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
- $S_1(M, S, R, H, Sc, P) = \sigma_{H \geq 100}(\text{Laptop}(M, S, R, H, Sc, P))$   
 $S_2(Ma, M, T, S, R, H, Sc, P) = \text{Product}(Ma, M, T) \bowtie S_1(M, S, R, H, Sc, P)$   
 $R(\text{maker}) = \pi_{Ma}(S_2(Ma, M, T, S, R, H, Sc, P))$
  - $R(\text{maker}) = \pi_{\text{maker}}(\text{Product} \bowtie \sigma_{hd \geq 100}(\text{Laptop}))$

# Relational Algebra Expressions $\leftrightarrow$ Query Execution Trees

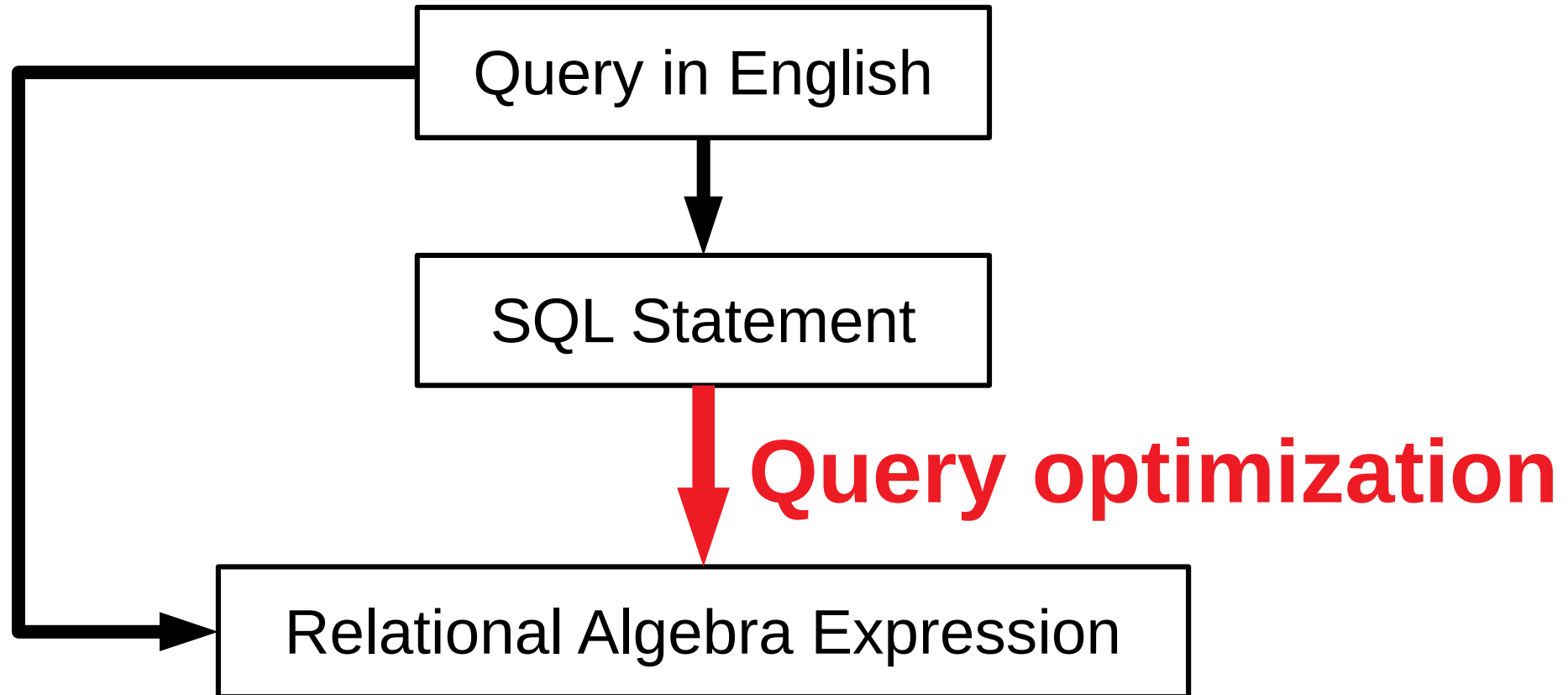
- $S_1(M, S, R, H, Sc, P) =$   
 $\sigma_{H \geq 100}(\text{Laptop}(M, S, R, H, Sc, P))$
- $S_2(Ma, M, T, S, R, H, Sc, P) =$   
 $\text{Product}(Ma, M, T) \bowtie$   
 $S_1(M, S, R, H, Sc, P)$
- $R(\text{maker}) =$   
 $\pi_{Ma}(S_2(Ma, M, T, S, R, H, Sc, P))$
- $R(\text{maker}) = \pi_{\text{maker}}(\text{Product} \bowtie$   
 $\sigma_{hd \geq 100}(\text{Laptop}))$



# Relational Algebra $\leftrightarrow$ SQL

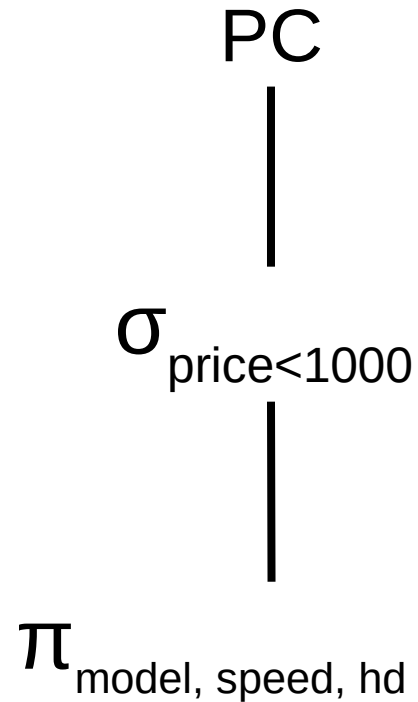
- SELECT  $\leftrightarrow$  Projection  $\pi$
- FROM  $\leftrightarrow$  Input tables
- WHERE  $\leftrightarrow$  Selection  $\sigma$ , Join predicates
- DISTINCT  $\leftrightarrow$  Duplicate elimination  $\delta$
- ORDER BY  $\leftrightarrow$  Sorting  $\tau$
- GROUP BY  $\leftrightarrow$  GroupBy aggregations  $\gamma$
- UNION, INTERSECT, EXCEPT  $\leftrightarrow$  Set operations  $\cup, \cap, -$
- JOIN  $\leftrightarrow$  Join

# From Queries (Through SQL) To Relational Algebra Expressions



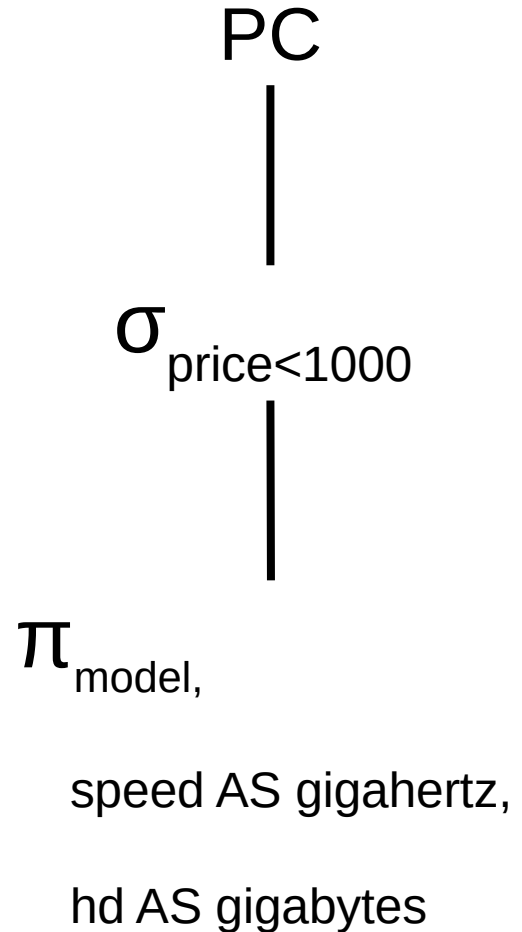
## 6.1.3 a)

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



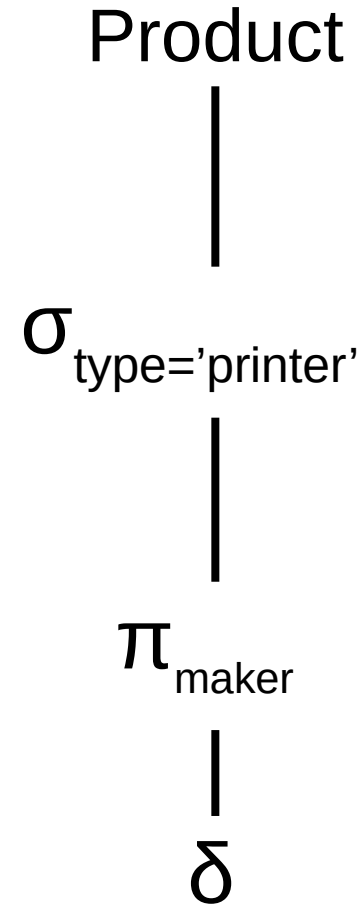
## 6.1.3 b)

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



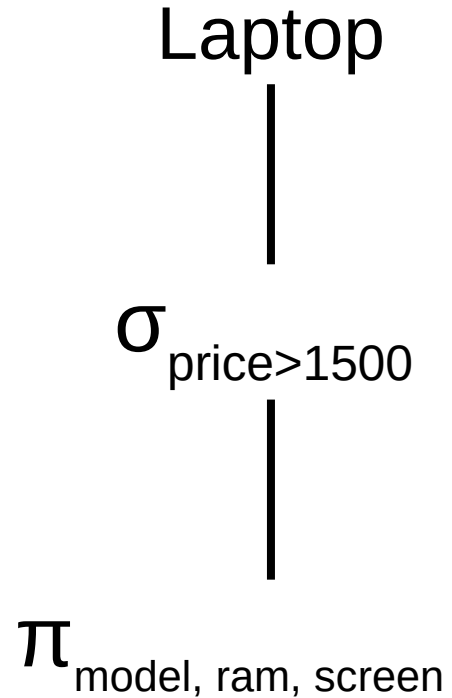
## 6.1.3 c)

select distinct maker  
from product  
where type = 'printer'



## 6.1.3 d)

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



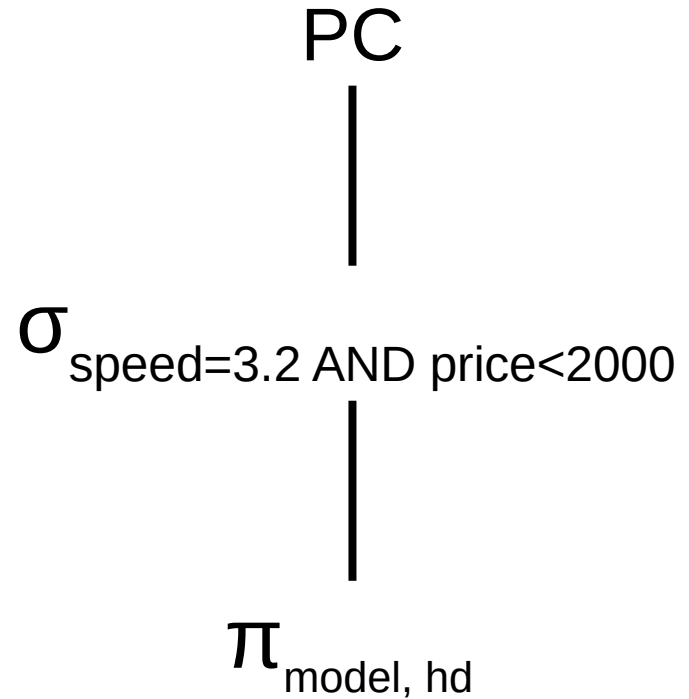
## 6.1.3 e)

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

Printer  
|  
 $\sigma_{\text{color=true}}$

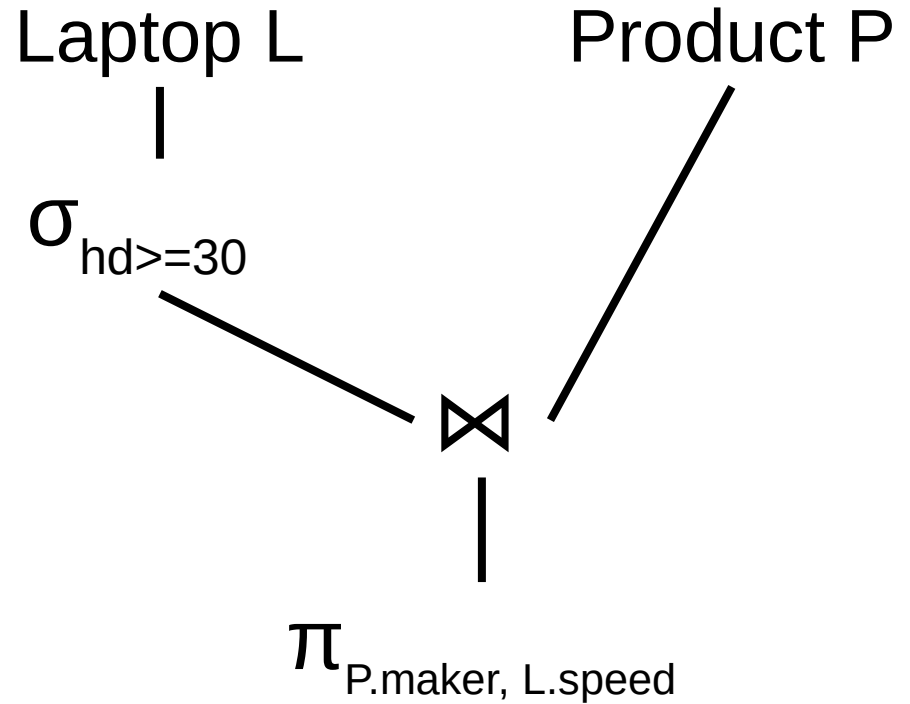
## 6.1.3 f)

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

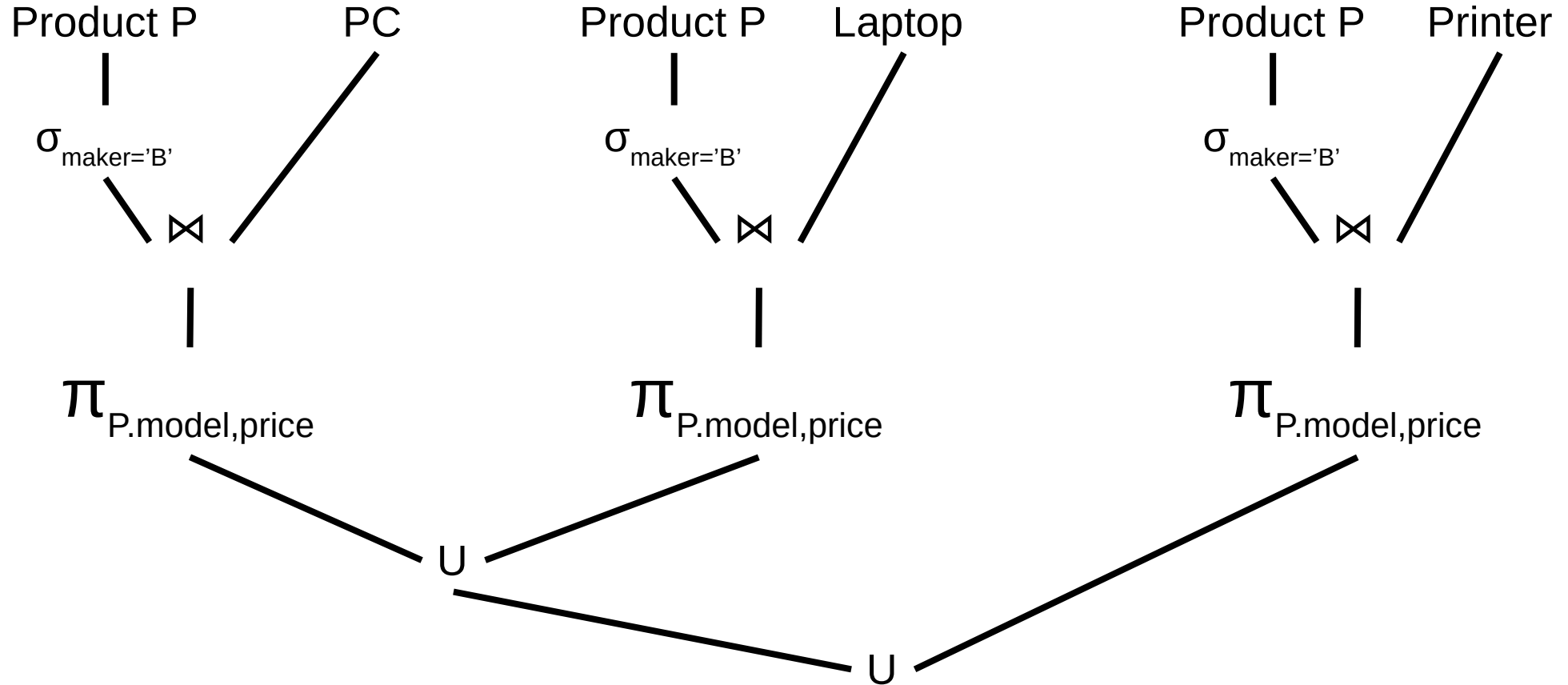


## 6.2.2 a)

select P.maker, L.speed  
from Product P, Laptop L  
where P.model = L.model  
AND hd >= 30

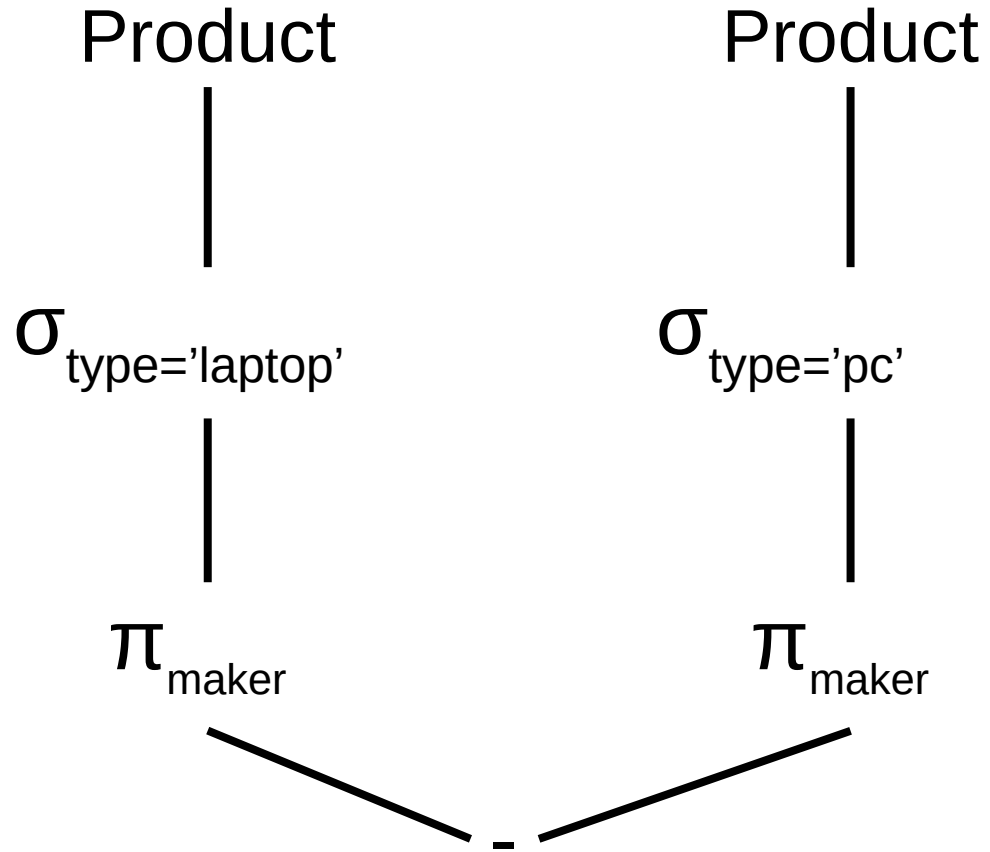


## 6.2.2 b)



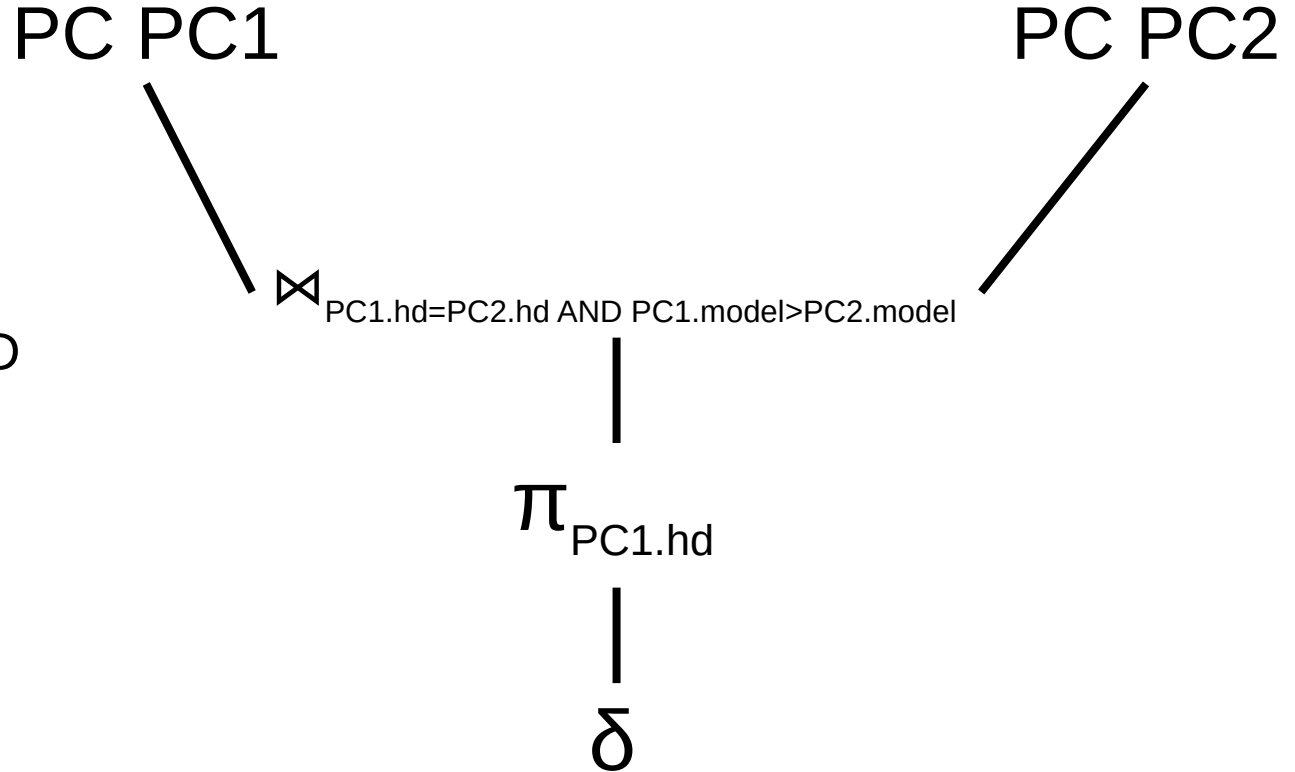
## 6.2.2 c)

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



## 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



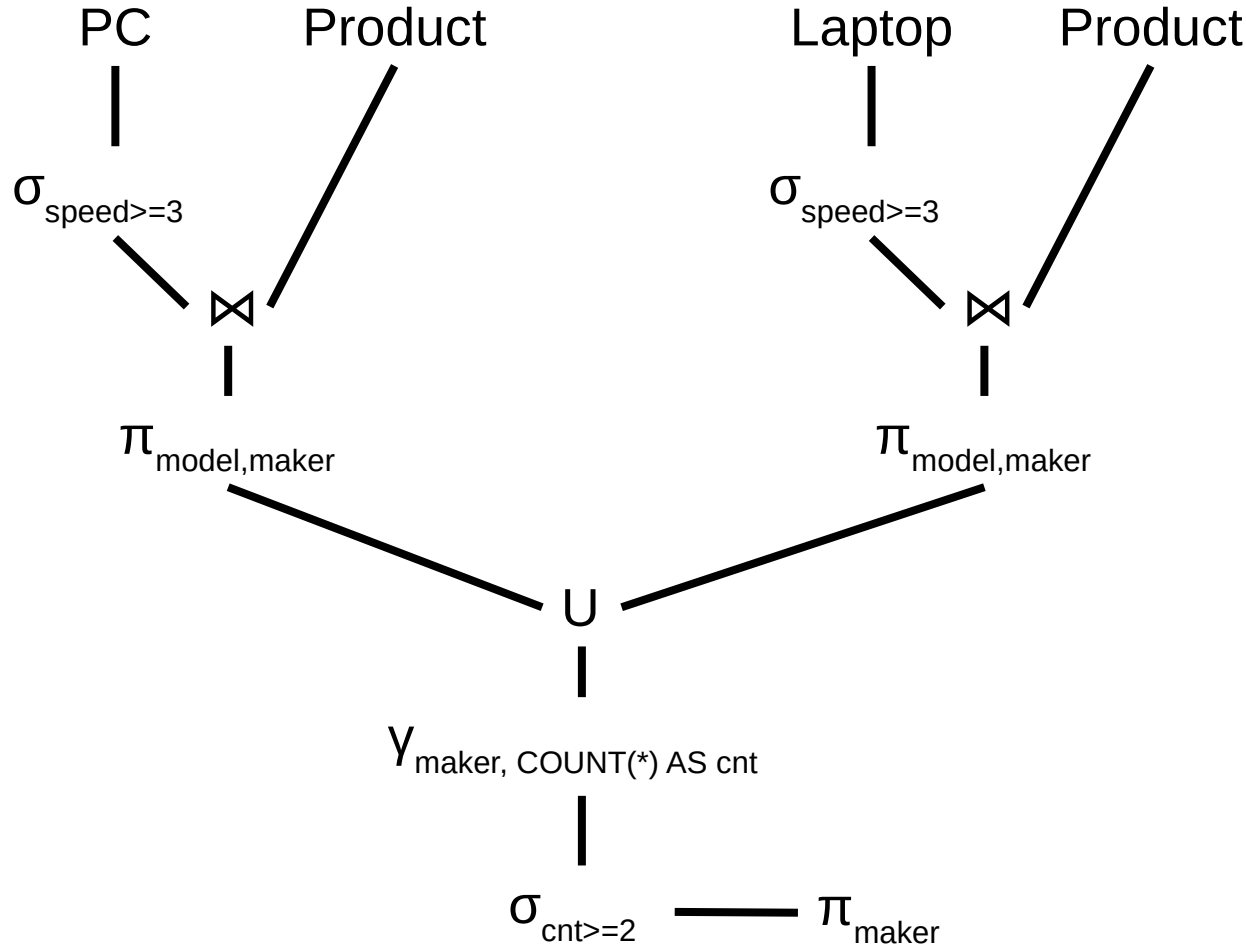
PC1.speed=PC2.speed AND  
PC.ram=PC2.ram AND  
PC1.model<PC2.model

$\pi$

PC1.model AS model\_1,

PC2.model AS model\_2

## 6.2.2 f)



- $S_1(\text{model, maker}) = \pi_{\text{model, maker}}(\text{Product} \bowtie \sigma_{\text{speed} \geq 3}(\text{PC}))$
- $S_2(\text{model, maker}) = \pi_{\text{model, maker}}(\text{Product} \bowtie \sigma_{\text{speed} \geq 3}(\text{Laptop}))$
- $S_3(\text{model, maker}) = S_1 \cup S_2$
- $S_4(\text{maker, cnt}) = \gamma_{\text{maker, COUNT(*) AS cnt}}(S_3)$
- $S_5(\text{maker, cnt}) = \sigma_{\text{cnt} \geq 2}(S_4)$
- $R(\text{maker}) = \pi_{\text{maker}}(S_5)$

# Relational Algebra

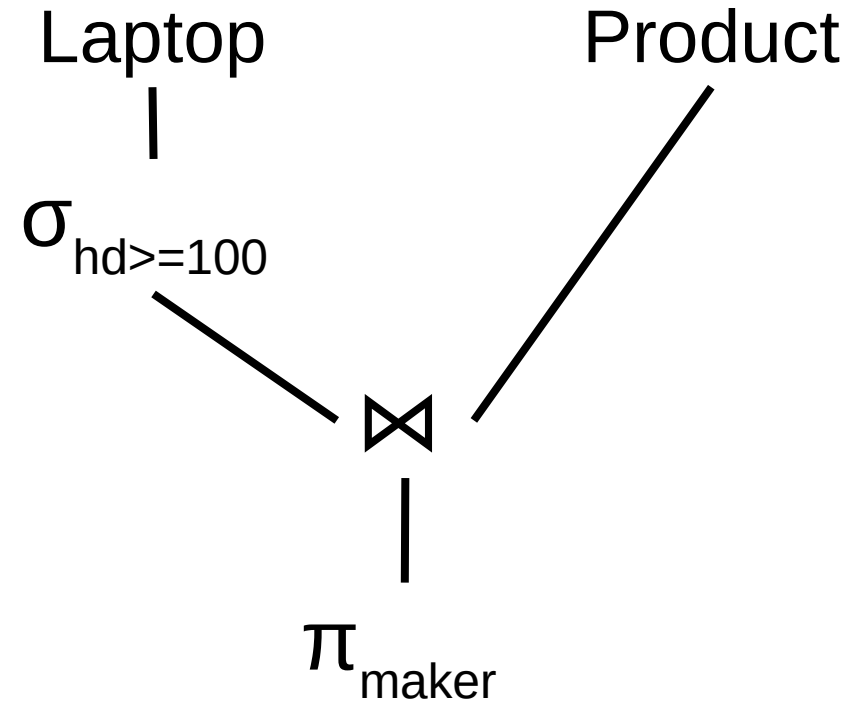
## Query Execution Tree Examples

# Relational Algebra Operators

- Projection  $\pi$
- Selection  $\sigma$
- Duplicate elimination  $\delta$
- Sorting  $\tau$
- GroupBy aggregations  $\gamma$
- Set operations  $\cup, \cap, -$
- Product  $\times$
- Join  $\bowtie$
- 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 $\leftrightarrow$ Query Execution Trees

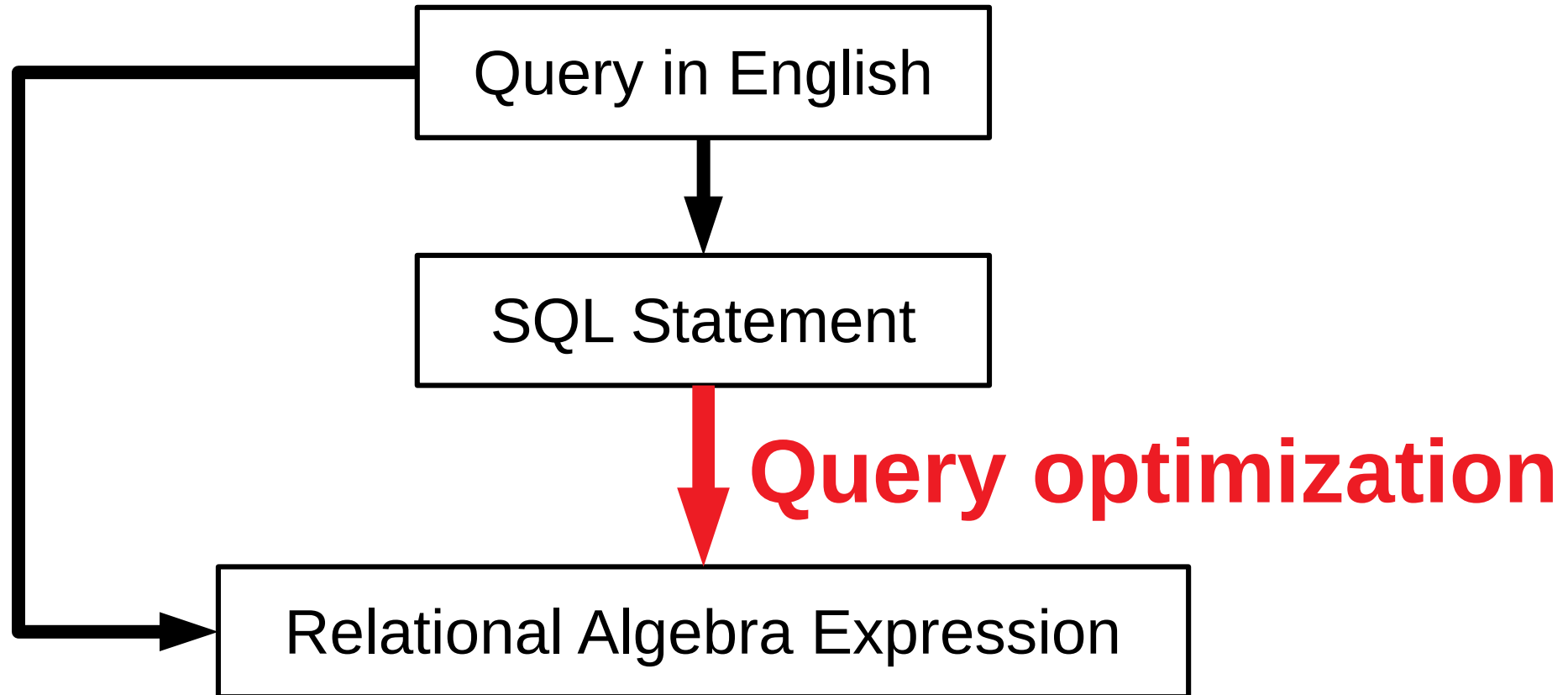
- $S_1(M, S, R, H, Sc, P) =$   
 $\sigma_{H \geq 100}(\text{Laptop}(M, S, R, H, Sc, P))$
- $S_2(Ma, M, T, S, R, H, Sc, P) =$   
 $\text{Product}(Ma, M, T) \bowtie$   
 $S_1(M, S, R, H, Sc, P)$
- $R(\text{maker}) =$   
 $\pi_{Ma}(S_2(Ma, M, T, S, R, H, Sc, P))$
- $R(\text{maker}) = \pi_{\text{maker}}(\text{Product} \bowtie$   
 $\sigma_{hd \geq 100}(\text{Laptop}))$



# Relational Algebra $\leftrightarrow$ SQL

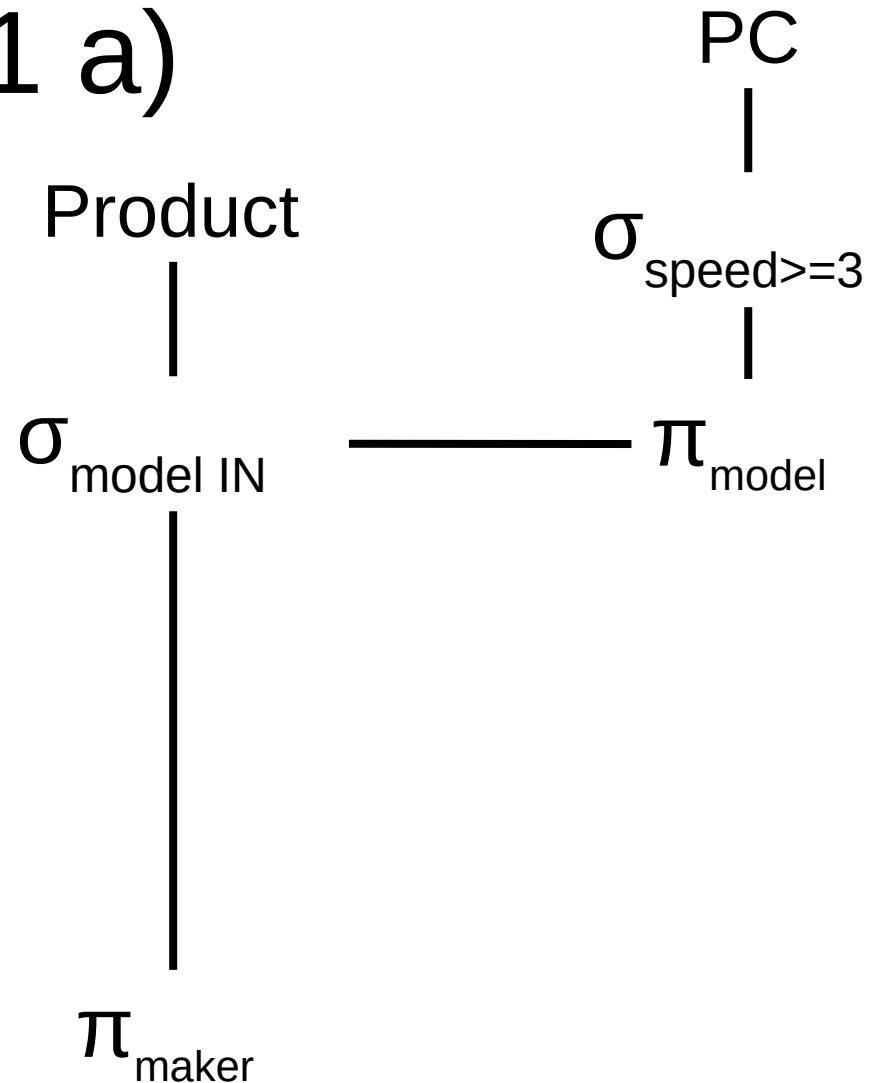
- SELECT  $\leftrightarrow$  Projection  $\pi$
- FROM  $\leftrightarrow$  Input tables
- WHERE  $\leftrightarrow$  Selection  $\sigma$ , Join predicates
- DISTINCT  $\leftrightarrow$  Duplicate elimination  $\delta$
- ORDER BY  $\leftrightarrow$  Sorting  $\tau$
- GROUP BY  $\leftrightarrow$  GroupBy aggregations  $\gamma$
- UNION, INTERSECT, EXCEPT  $\leftrightarrow$  Set operations  $\cup, \cap, -$
- JOIN  $\leftrightarrow$  Join

# From Queries (Through SQL) To Relational Algebra Expressions



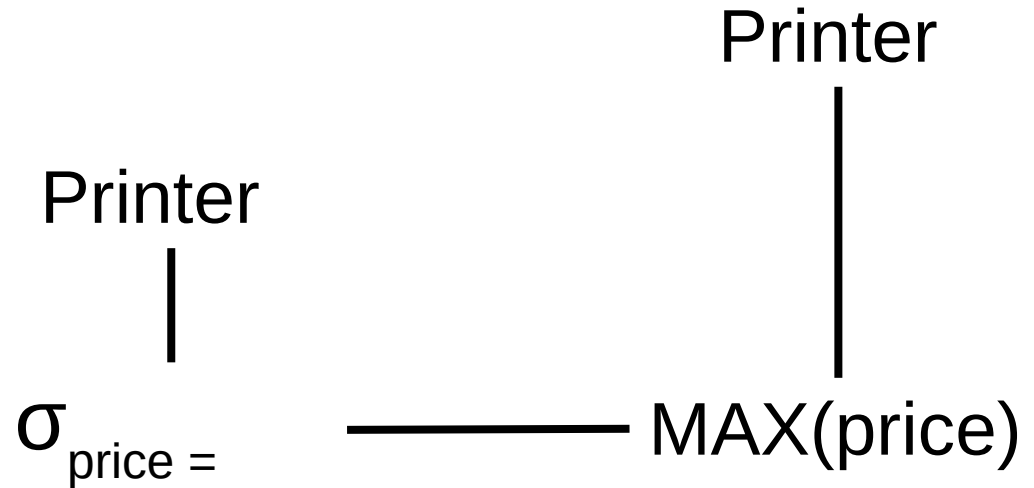
## 6.3.1 a)

select maker  
from Product  
where model in  
    (select model  
      from PC  
      where speed >= 3)



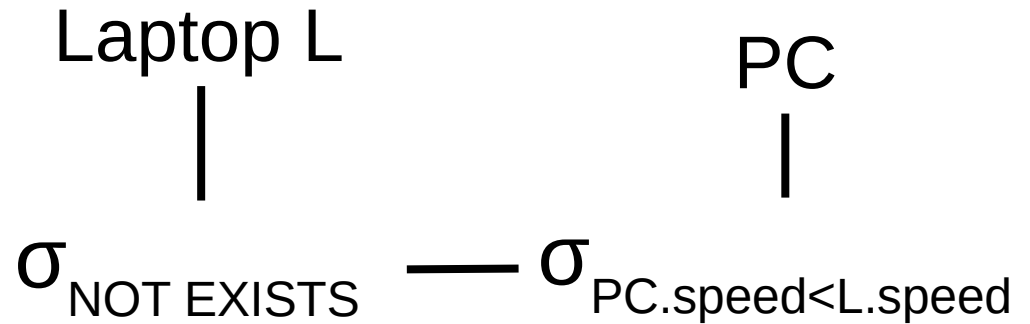
## 6.3.1 b)

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

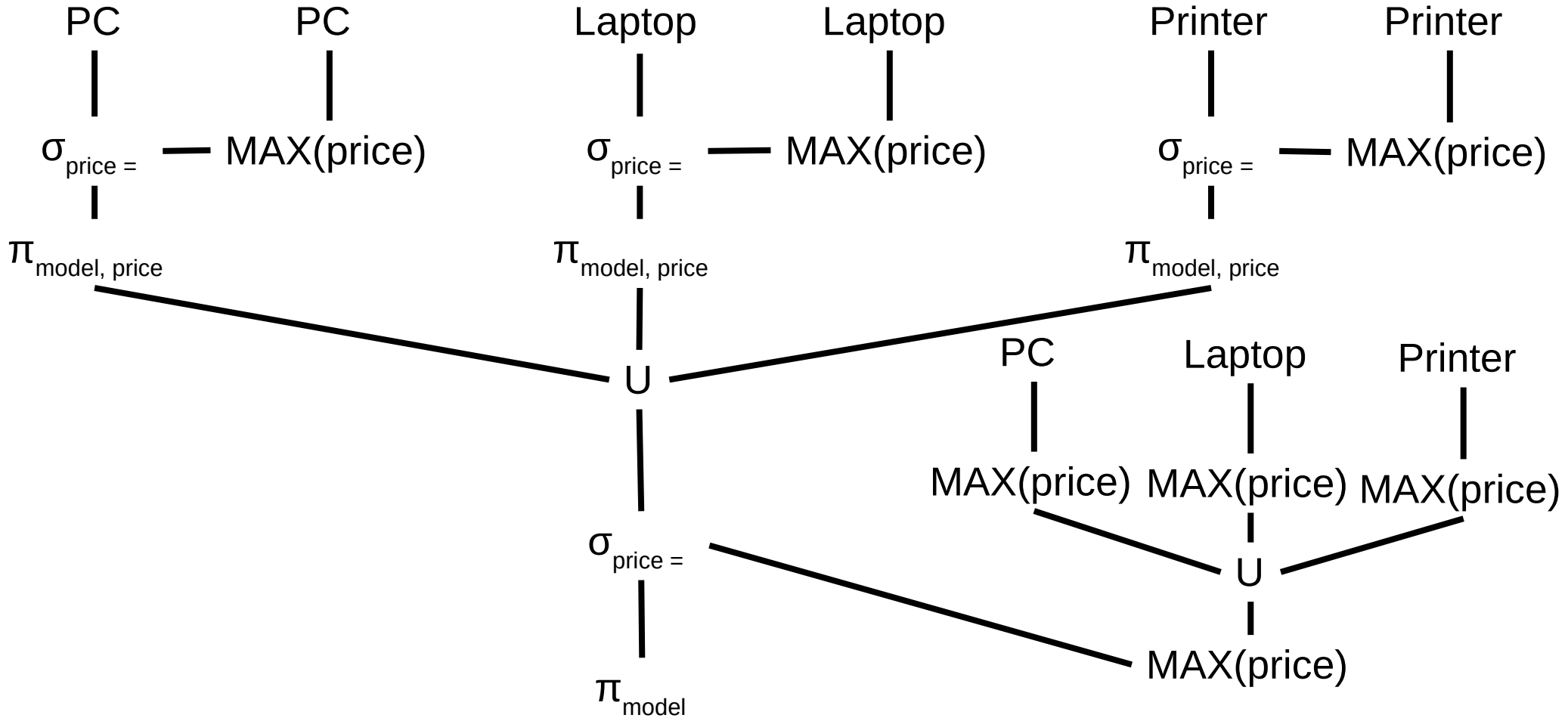


## 6.3.1 c)

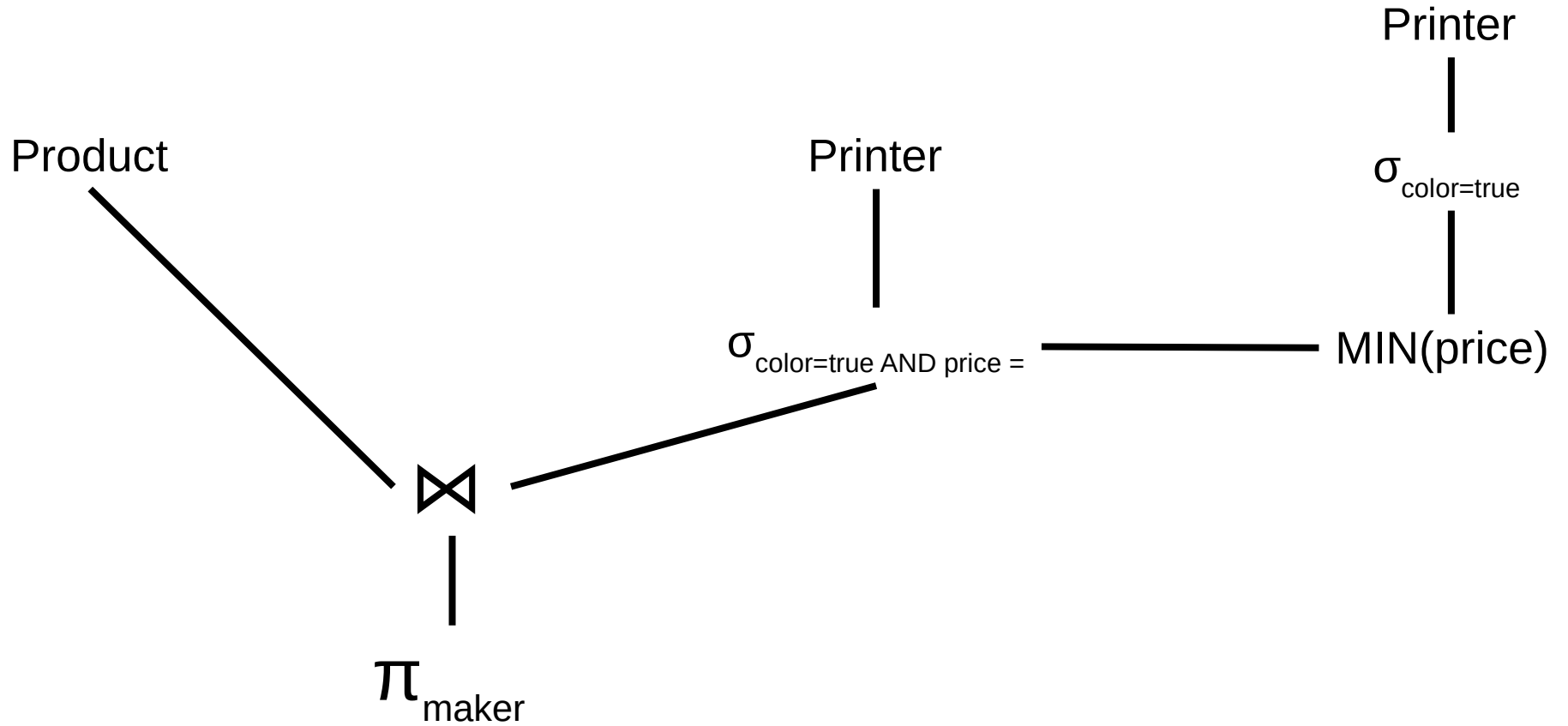
```
select *
from Laptop L
where not exists
 (select *
 from PC
 where PC.speed < L.speed)
```



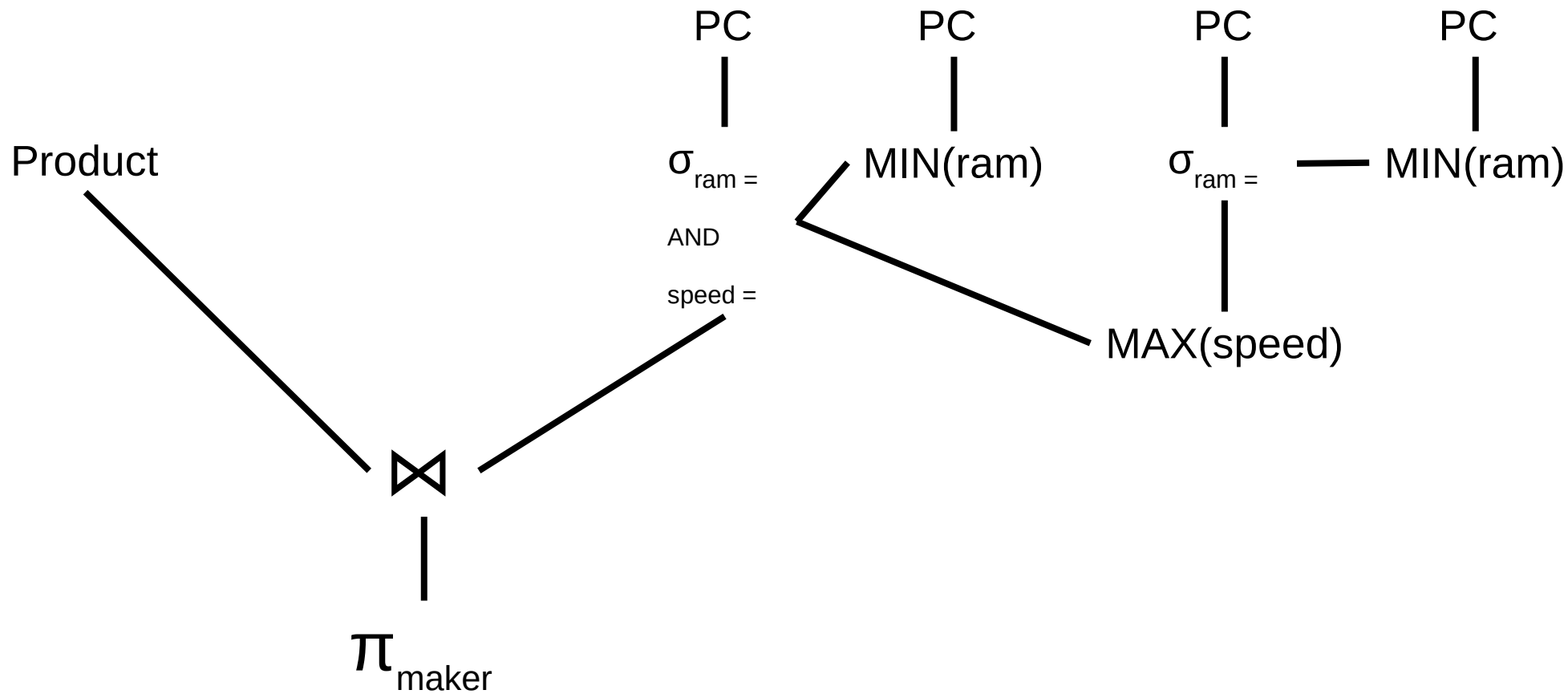
# 6.3.1 d)



# 6.3.1 e)



# 6.3.1 f)



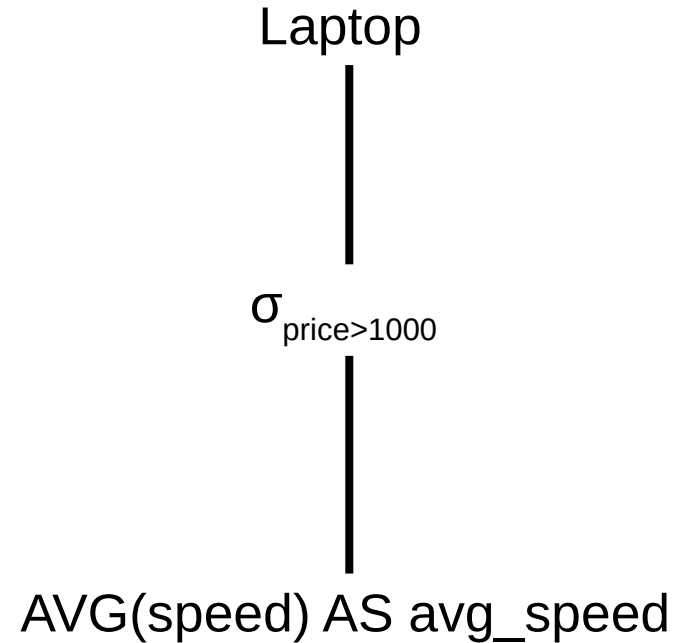
## 6.4.6 a)

```
select avg(speed) as
avg_speed
from pc
```

PC  
|  
AVG(speed) AS avg\_speed

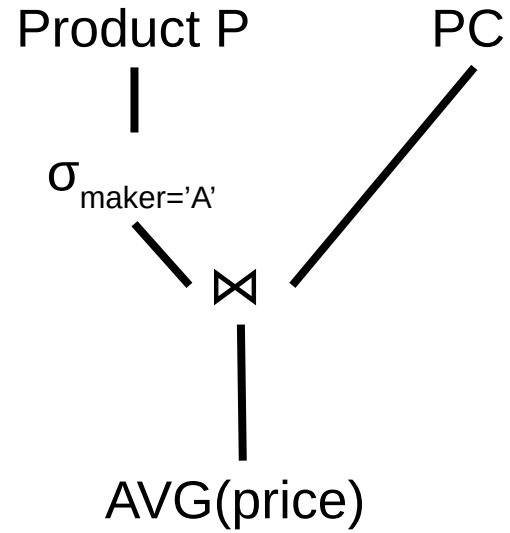
## 6.4.6 b)

```
select avg(speed) as
avg_speed
from laptop
where price > 1000
```

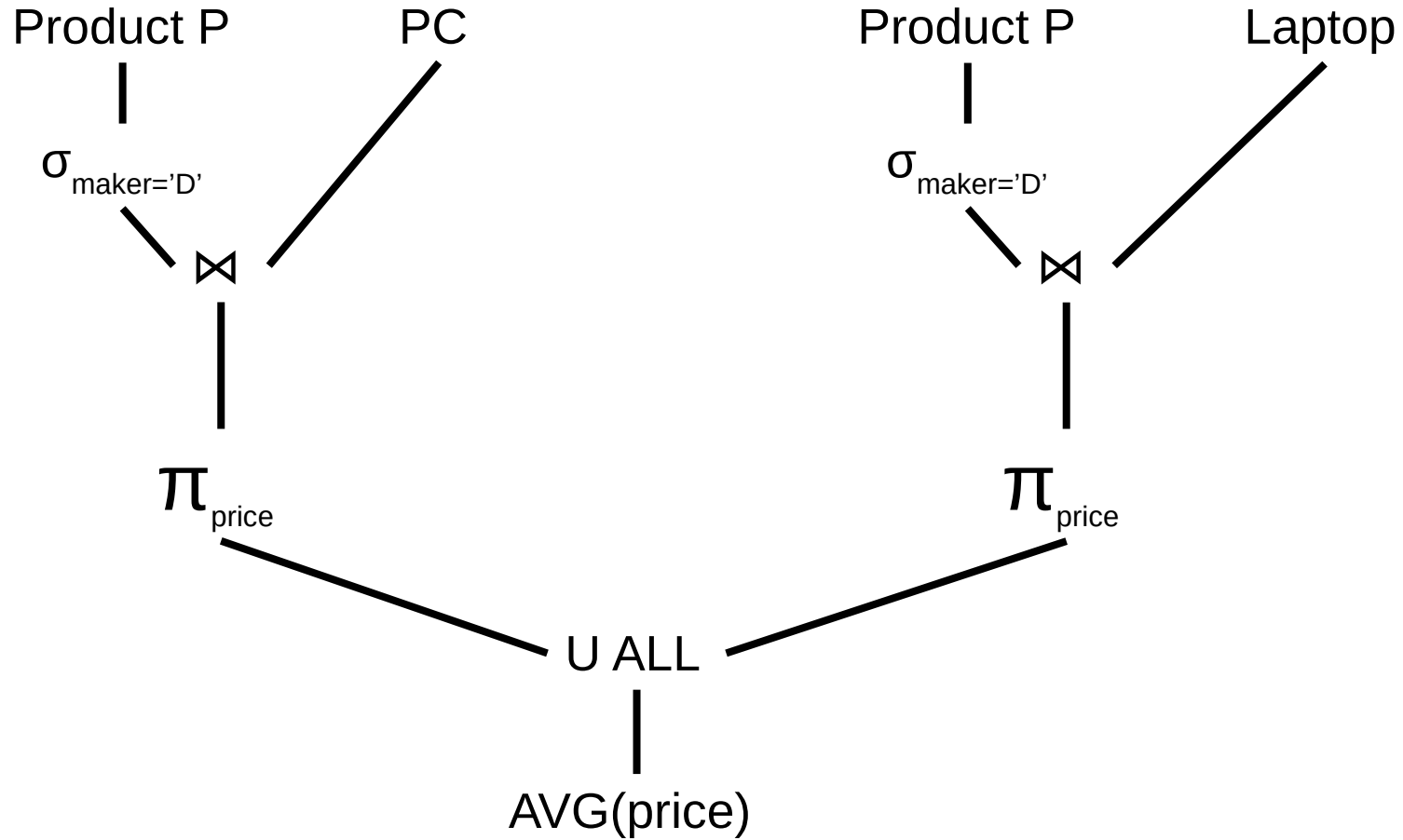


## 6.4.6 c)

select avg(price)  
from Product P, PC  
where P.model = PC.model AND  
P.maker = 'A'



# 6.4.6 d)



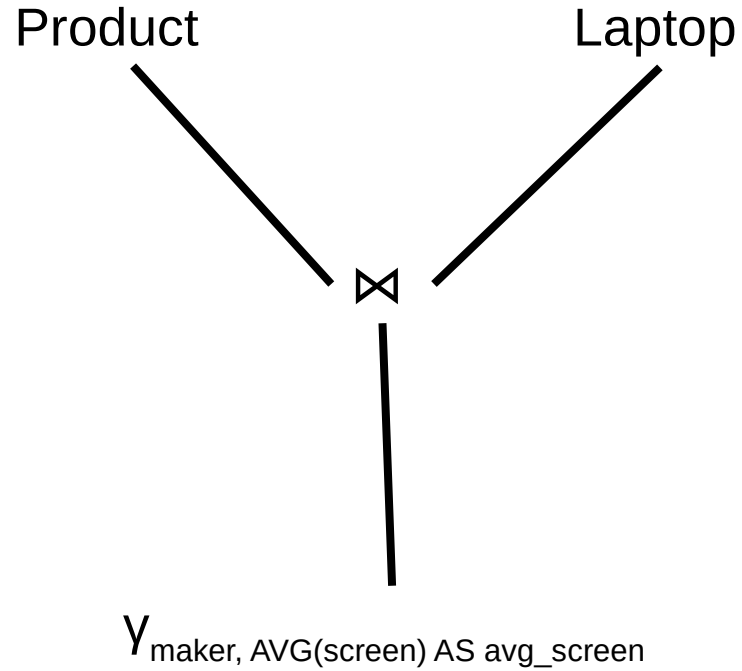
## 6.4.6 e)

```
select speed, avg(price) as
avg_price
from pc
group by speed
```

PC  
|  
Y<sub>speed, AVG(price) AS avg\_price</sub>

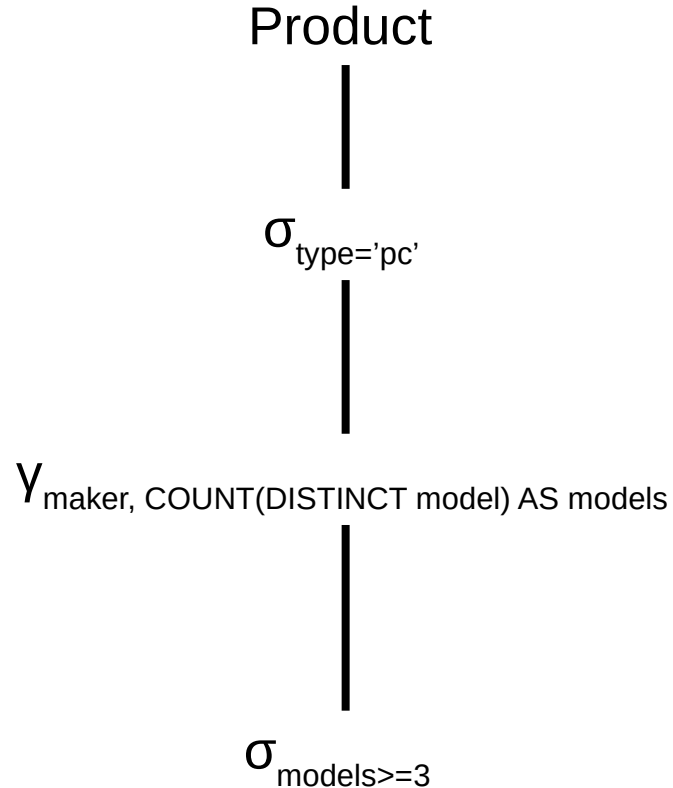
## 6.4.6 f)

```
select maker, avg(screen) as
avg_screen
from Product P, Laptop L
where P.model = L.model
group by maker
```



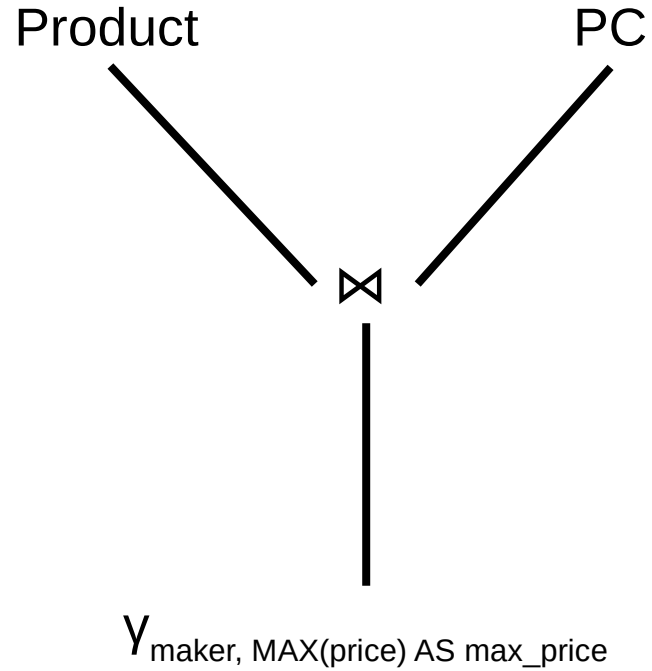
## 6.4.6 g)

select maker, count (distinct  
model) as models  
from product  
where type = 'pc'  
group by maker  
having models >= 3



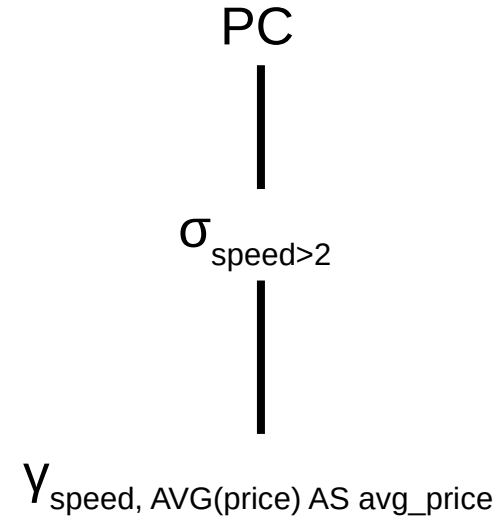
## 6.4.6 h)

```
select maker, max(price) as
max_price
from Product P, PC
where P.model = PC.model
group by maker
```



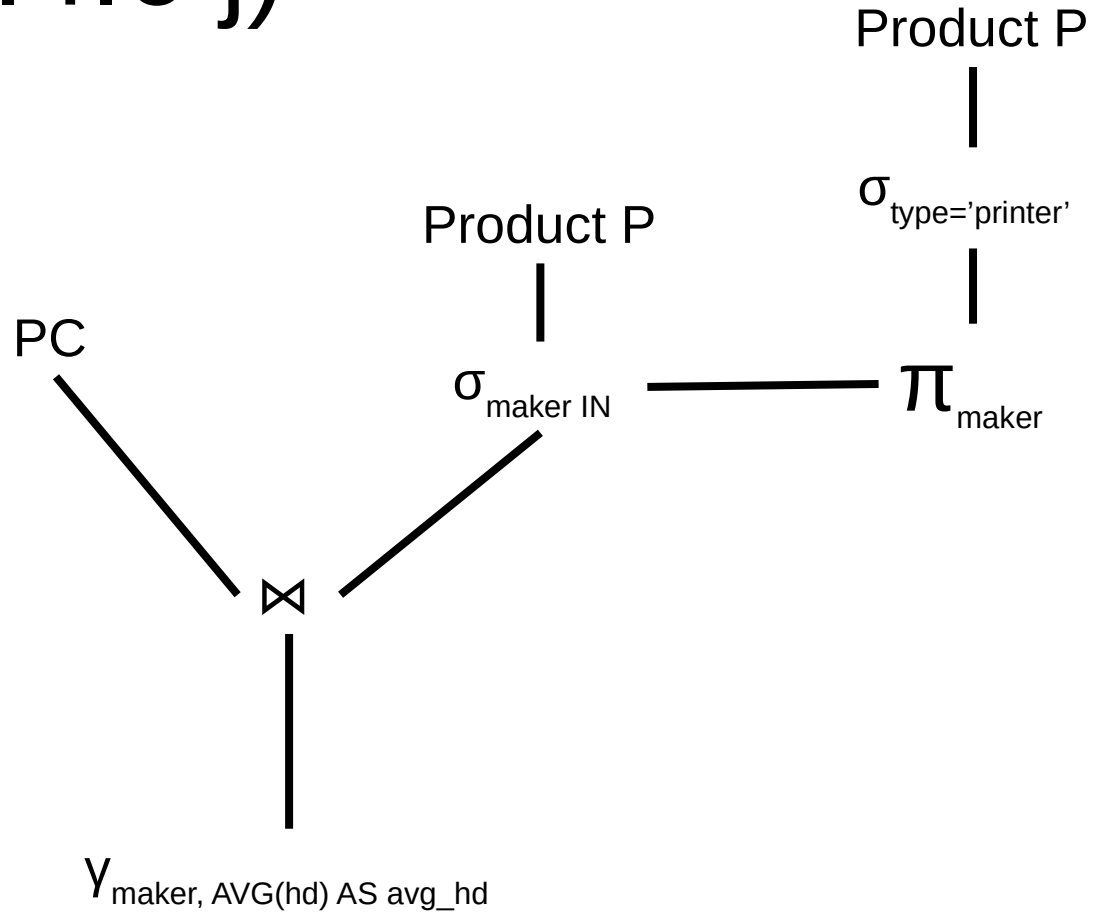
## 6.4.6 i)

```
select speed, avg(price)
as avg_price
from pc
where speed > 2
group by speed
```



## 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](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](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](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); --""")`

Indexes

# Useful SQLite Commands

- *.eqp on|off*
  - Show execution plan for SQL query
- *.timer on|off*
  - Show execution time for SQL statement
- *.output file*
  - Print SQL statement output to *file*
- *.read file*
  - Execute SQL statements from *file*

# Query Types

- Full table scan
  - select l\_orderkey from lineitem
- Point query
  - select l\_orderkey from lineitem where **l\_quantity = 10**
- Range query
  - select l\_orderkey from lineitem where **l\_quantity < 10**
  - select l\_orderkey from lineitem where **l\_quantity >= 10 and l\_quantity <= 20**

# Query Execution Plans

- Full table scan
  - select l\_orderkey from lineitem
    - 60,175 tuples
    - **--SCAN TABLE lineitem**
- Point query
  - select l\_orderkey from lineitem where l\_quantity = 10
    - 1,182 tuples
    - **--SCAN TABLE lineitem**
- Range query
  - select l\_orderkey from lineitem where l\_quantity < 10
    - 10,816 tuples
    - **--SCAN TABLE lineitem**
  - select l\_orderkey from lineitem where l\_quantity >= 10 and l\_quantity <= 20
    - 13,071 tuples
    - **--SCAN TABLE lineitem**

# Indexes

- Query time is proportional with the number of tuples accessed by the query
  - More tuples accessed → larger query time
- Reduce number of accessed tuples by creating a copy of an attribute and sort it increasingly
  - Binary search on sorted data
  - Pointer to the complete tuple
- Trade-off space for query time

# Indexes in SQLite

- **CREATE INDEX *lineitem\_idx\_l\_quantity* ON lineitem(l\_quantity)**
- **DROP INDEX *lineitem\_idx\_l\_quantity***
- Database server decides when and how to use indexes for query processing
  - User cannot control index usage

# Query Execution Plans with Indexes

- Full table scan
  - select l\_orderkey from lineitem
    - 60,175 tuples
    - **--SCAN TABLE lineitem**
- Point query
  - select l\_orderkey from lineitem where l\_quantity = 10
    - 1,182 tuples
    - **--SEARCH TABLE lineitem USING INDEX ind\_l\_quantity (l\_quantity=?)**
- Range query
  - select l\_orderkey from lineitem where l\_quantity < 10
    - 10,816 tuples
    - **--SEARCH TABLE lineitem USING INDEX ind\_l\_quantity (l\_quantity<?)**
  - select l\_orderkey from lineitem where l\_quantity >= 10 and l\_quantity <= 20
    - 13,071 tuples
    - **--SEARCH TABLE lineitem USING INDEX ind\_l\_quantity (l\_quantity>? AND l\_quantity<?)**

# Database Size Increase

- `ls -la` command
- Before CREATE INDEX
  - data/tpch.sqlite: **11288576 bytes**
- After CREATE INDEX
  - data/tpch.sqlite: **11862016 bytes**
  - Increase of **573440 bytes**

# Query Execution Time (Decrease)

- select l\_orderkey from lineitem
  - **--SCAN TABLE lineitem** → 14 ms
- select l\_orderkey from lineitem where l\_quantity = 10
  - **--SCAN TABLE lineitem** → 6 ms
  - **--SEARCH TABLE lineitem USING INDEX ind\_l\_quantity (l\_quantity=?)** → 3 ms
- select l\_orderkey from lineitem where l\_quantity < 10
  - **--SCAN TABLE lineitem** → 8 ms
  - **--SEARCH TABLE lineitem USING INDEX ind\_l\_quantity (l\_quantity<?)** → 16 ms
- select l\_orderkey from lineitem where l\_quantity >= 10 and l\_quantity <= 20
  - **--SCAN TABLE lineitem** → 8 ms
  - **--SEARCH TABLE lineitem USING INDEX ind\_l\_quantity (l\_quantity>? AND l\_quantity<?)** → 19 ms

# INSERT Execution Time Increase

- **insert into lineitem (select \* from lineitem)**
  - No index → **120 ms**
  - Index → **156 ms**

# Index Recommendation

- SQLite recommends indexes for a query based on data and existing indexes

*.expert*

*select l\_orderkey from lineitem where l\_quantity = 10*

- CREATE INDEX lineitem\_idx\_l\_quantity ON lineitem(l\_quantity)

# Indexes Summary

- Increase in storage space
- Decrease in query execution time
  - Only for very selective queries with result tuples very small compared to table tuples (1,182/60,175)
- Increase in MODIFICATION (I/U/D) execution time
  - Modify both the table and the index

Views

# Useful SQLite Commands

- *.eqp on|off*
  - Show execution plan for SQL query
- *.schema*
  - Show CREATE [TABLE & VIEW] statements
- *.tables*
  - Show tables and views

# Virtual Views

- The equivalent of macros and inline functions from C/C++
  - #define constructs, functions in header files
  - Give a name to a code segment rather than copy the code in multiple places
  - The copying is done automatically by the macro processor or compiler without the programmer intervention
- Improve code organization and readability
- No performance benefit

# Virtual View Definition in SQL

```
CREATE VIEW Printer_Maker(model, color,
type, price, maker) AS
```

```
 select Pr.model, Pr.color, Pr.type, price, maker
```

```
 from Printer Pr, Product P
```

```
 where Pr.model = P.model
```

```
DROP VIEW Printer_Maker
```

# View Usage in SQL Queries

```
select *
from Printer_Maker
```

```
select *
from
 (select Pr.model, Pr.color,
 Pr.type, price, maker
 from Printer Pr, Product P
 where Pr.model =
 P.model) Printer_Maker
```

# View Usage in SQL Queries

```
select distinct maker
from product p,
Printer pr
where p.model =
pr.model
```

```
and color = true
and price < 200
```

```
select distinct maker
from Printer_Maker
where color = true
and price < 200
```

# Virtual Views

- The SQL query in the view definition is not evaluated, it is simply given a name *Printer\_Maker*
- In SQLite, *CREATE VIEW* is added to the existing database tables
  - *.tables* or *.schema*
  - There is no other change to the database
- The query execution plans with and without the view are exactly the same
  - *.eqp on*

# Modification Operations on Tables in the Virtual View Definition

- DROP TABLE Printer
  - View *Printer\_Maker* becomes invalid
- INSERT/DELETE/UPDATE on Printer
  - All modification operations are immediately reflected in the view since the query in the view is re-evaluated every time it is included in a query
  - Exactly the same behavior as for tables

# Modification Operations on Virtual Views

INSERT INTO Printer\_Maker(model, color, type, price, maker)

VALUES(3108, false, 'laser', 169, 'A')

- INSERT INTO Printer(model, color, type, price)  
VALUES(3108, false, 'laser', 169)
- INSERT INTO Product(model, type, maker)  
VALUES(3108, 'printer', 'A')

# Modification Operations on Virtual Views

```
CREATE VIEW Prod_Printer(model, maker) AS
 SELECT model, maker
 FROM Product
 WHERE type = 'printer'
```

```
INSERT INTO Prod_Printer(model, maker)
VALUES(3108, 'A')
```

- INSERT INTO Product(model, type, maker)  
VALUES(3108, NULL, 'A')
- SELECT \* FROM Prod\_Printer
  - **(3108, 'A') is not in the result**

# Modification Operations on Virtual Views

- **Not supported in SQLite**
- SQL standard defines **UPDATABLE VIEWS**

```
CREATE VIEW Prod_Printer(model, maker, type)
AS
```

```
 SELECT model, maker, type
```

```
 FROM Product
```

```
 WHERE type = 'printer'
```

# Materialized Views

- Query result caching (materialization) into a table
- Use in queries exactly as tables or views
- Avoid recomputation by returning result directly
- Related to memoization from dynamic programming
- **Improve query performance**

# Materialized View Definition in SQL

## **CREATE MATERIALIZED VIEW**

Printer\_Maker\_M(model, color, type, price,  
maker) AS

select Pr.model, Pr.color, Pr.type, price, maker  
from Printer Pr, Product P  
where Pr.model = P.model

**DROP MATERIALIZED VIEW** Printer\_Maker\_M

- **Not supported in SQLite**

# Simulate Materialized Views in SQLite

## **CREATE MATERIALIZED VIEW**

```
Printer_Maker_M(model,
color, type, price, maker)
AS
```

```
select Pr.model, Pr.color,
Pr.type, price, maker
from Printer Pr, Product P
where Pr.model = P.model
```

## • **CREATE TABLE**

```
Printer_Maker_M(model,
color, type, price, maker)
```

## • **INSERT INTO**

```
Printer_Maker_M
```

```
SELECT Pr.model,
Pr.color, Pr.type, price,
maker
```

```
from Printer Pr, Product P
where Pr.model = P.model
```

# Modification Operations on Materialized Views

- Since the materialized view is a table, I/U/D operations are straightforward
- View is not consistent with its definition anymore
- **For consistency, modification operations have to be propagated to the base tables in the view definition**
  - Same approach as for virtual views

# Materialized View Maintenance

- Materialized view is a separate table
  - Independent copy of data
- Modification operations on tables in the view definition have to be propagated to the view
  - **View is consistent with base tables**
- Naive materialized view maintenance
  - Complete reevaluation
  - DELETE FROM Printer\_Maker\_M
  - INSERT INTO Printer\_Maker\_M (SELECT ...)

# Incremental View Maintenance

- Minimize the number of tuples from the materialized view that get impacted by a modification operation on base tables
- Implemented for every I/U/D operation separately
  - Consider only modified tuples from base tables

# INSERT Product+Printer

- INSERT INTO Product(model, type, maker)  
VALUES(**3108**, 'printer', 'A')
- INSERT INTO Printer(model, color, type, price)  
VALUES(**3108**, false, 'laser', 169)
- **INSERT INTO Printer\_Maker\_M**  
*(SELECT Pr.model, Pr.color, Pr.type, price, maker  
from Printer Pr, Product P  
where Pr.model = P.model AND P.model = 3108)*

# DELETE Printer

- DELETE FROM Printer WHERE model < 3004
- **DELETE FROM Printer\_Maker\_M  
WHERE model < 3004**

# UPDATE Product

- UPDATE Product  
SET maker = 'A'  
WHERE maker = 'D'
- **UPDATE Printer\_Maker\_M**  
**SET maker = 'A'**  
**WHERE maker = 'D'**

# Views Summary

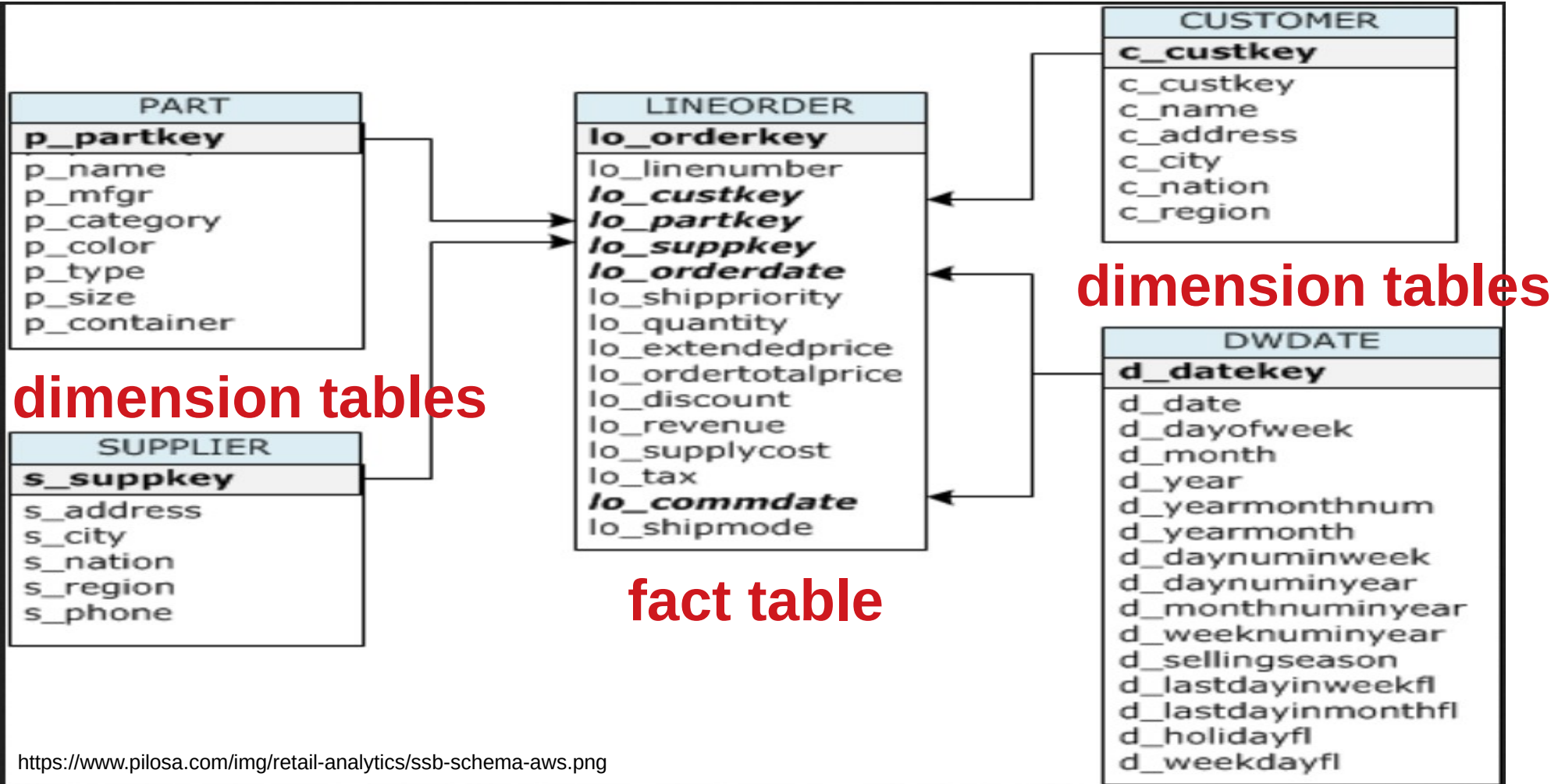
- Virtual views
  - Name for SELECT statement
  - Improve coding
  - No modification operations
  - No query execution performance improvement
- Materialized views
  - Save query result into a separate table
  - Query result caching
  - Always improve query execution performance
  - Incremental view maintenance

# On-Line Analytic Processing (OLAP) & Data Cubes

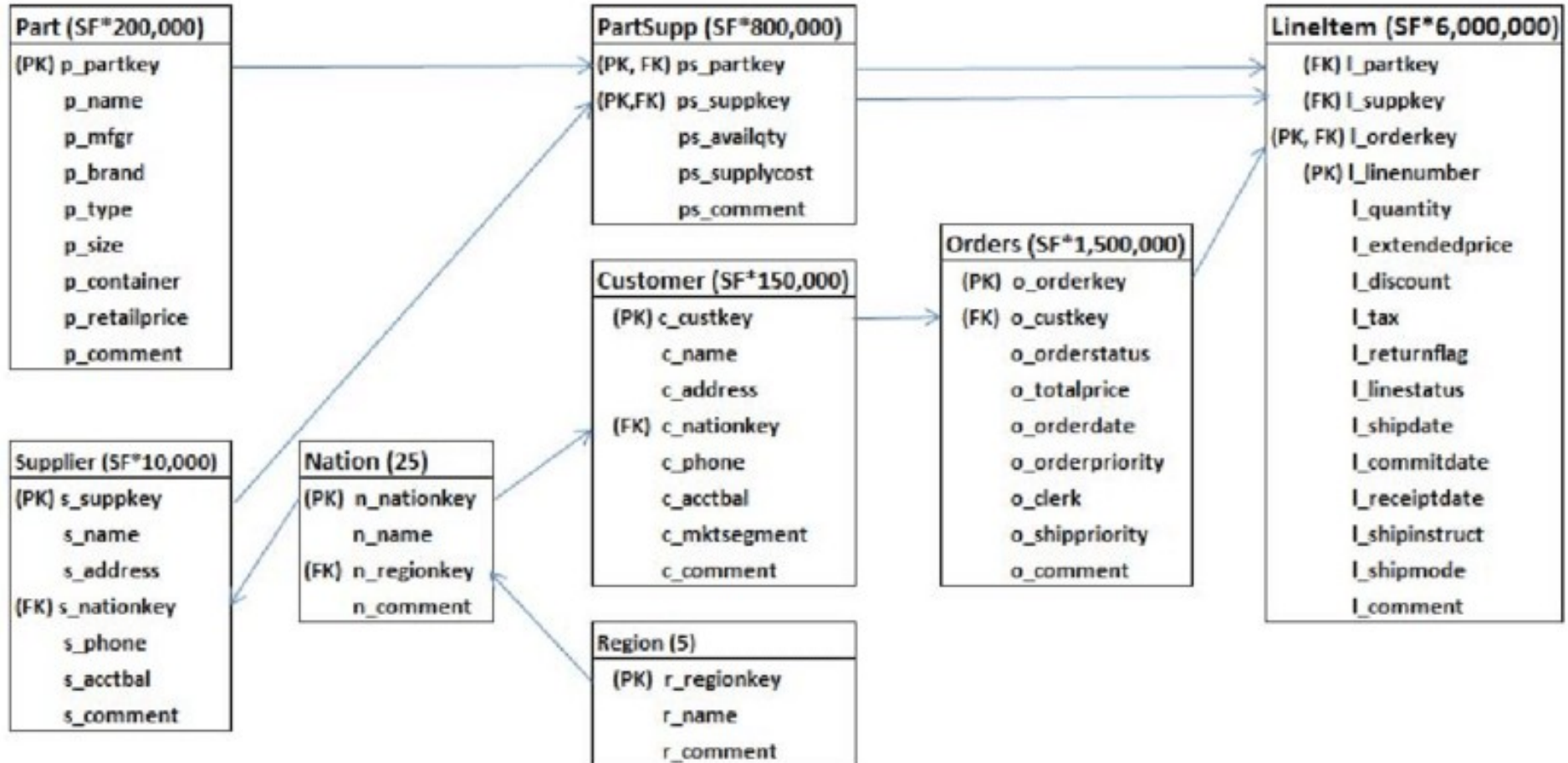
# OLAP vs. OLTP

- On-Line Analytic Processing (OLAP)
  - Decision-support over data warehouses
  - Highly complex queries with one or more aggregations
  - Examine large amounts of data even when the result is small
    - Queries in Lab 6 over TPC-H
- On-Line Transaction Processing (OLTP)
  - Modification operations (transactions)
  - Touch a tiny portion (one or a few tuples) of the database
    - Record a new order in TPC-H

# Star Schema



# TPC-H Schema (Snowflake)



# OLAP Query Example

```
select n_name, sum(o_totalprice) as tot_orders
```

```
from orders, customer, nation, region
```

```
where o_custkey = c_custkey
```

```
and c_nationkey = n_nationkey
```

```
and n_regionkey = r_regionkey
```

```
and o_orderdate >= '1996-01-01'
```

```
and o_orderdate < '1997-01-01'
```

```
and r_name = 'AMERICA'
```

```
group by n_name
```

```
order by tot_orders desc
```

CANADA|18482207.74

BRAZIL|15273545.8

UNITED STATES|11750866.68

ARGENTINA|11502493.16

PERU|9312955.18

# Slicing & Dicing OLAP Queries

```
SELECT <dicing attributes & aggregations>
FROM <fact table joined with dimension tables>
WHERE <slicing attributes>
GROUP BY <dicing attributes>
```

# Data Exploration with Drill-down and Roll-up

```
select n_name, sum(o_totalprice) as tot_orders
```

```
from orders, customer, nation, region
```

```
where o_custkey = c_custkey
```

```
and c_nationkey = n_nationkey
```

```
and n_regionkey = r_regionkey
```

```
and o_orderdate >= '1996-01-01'
```

```
and o_orderdate < '1997-01-01'
```

```
and r_name = 'AMERICA'
```

```
group by n_name
```

```
order by tot_orders desc
```

CANADA|18482207.74

BRAZIL|15273545.8

UNITED STATES|11750866.68

ARGENTINA|11502493.16

PERU|9312955.18

# Drill-down on Market Segment in US

```
select c_mktsegment, sum(o_totalprice) as tot_orders
from orders, customer, nation
where o_custkey = c_custkey
 and c_nationkey = n_nationkey
 and o_orderdate >= '1996-01-01'
 and o_orderdate < '1997-01-01'
 and n_name = 'UNITED STATES'
group by c_mktsegment
```

|            |            |
|------------|------------|
| AUTOMOBILE | 1764146.3  |
| BUILDING   | 3949798.52 |
| FURNITURE  | 2463719.39 |
| HOUSEHOLD  | 2807178.64 |
| MACHINERY  | 766023.83  |

# Drill-down on Month for BUILDING

```
select substr(o_orderdate, 6, 2) as month, sum(o_totalprice) as
tot_orders
from orders, customer, nation
where o_custkey = c_custkey
 and c_nationkey = n_nationkey
 and o_orderdate >= '1996-01-01'
 and o_orderdate < '1997-01-01'
 and n_name = 'UNITED STATES'
 and c_mktsegment = 'BUILDING'
group by month
```

|    |           |
|----|-----------|
| 01 | 307934.57 |
| 04 | 449200.42 |
| 05 | 504249.66 |
| 06 | 535603.54 |
| 07 | 197825.34 |
| 08 | 133971.12 |
| 09 | 719143.56 |
| 10 | 446284.43 |
| 11 | 401000.0  |
| 12 | 254585.88 |

# Roll-up on Month

```
select substr(o_orderdate, 6, 2) as month, sum(o_totalprice)
as tot_orders
```

```
from orders, customer, nation
```

```
where o_custkey = c_custkey
```

```
and c_nationkey = n_nationkey
```

```
and o_orderdate >= '1996-01-01'
```

```
and o_orderdate < '1997-01-01'
```

```
and n_name = 'UNITED STATES'
```

```
group by month
```

01|763247.39

02|589382.5

03|41703.87

04|1424955.06

05|1107433.27

06|1239444.19

07|992346.92

08|848086.1

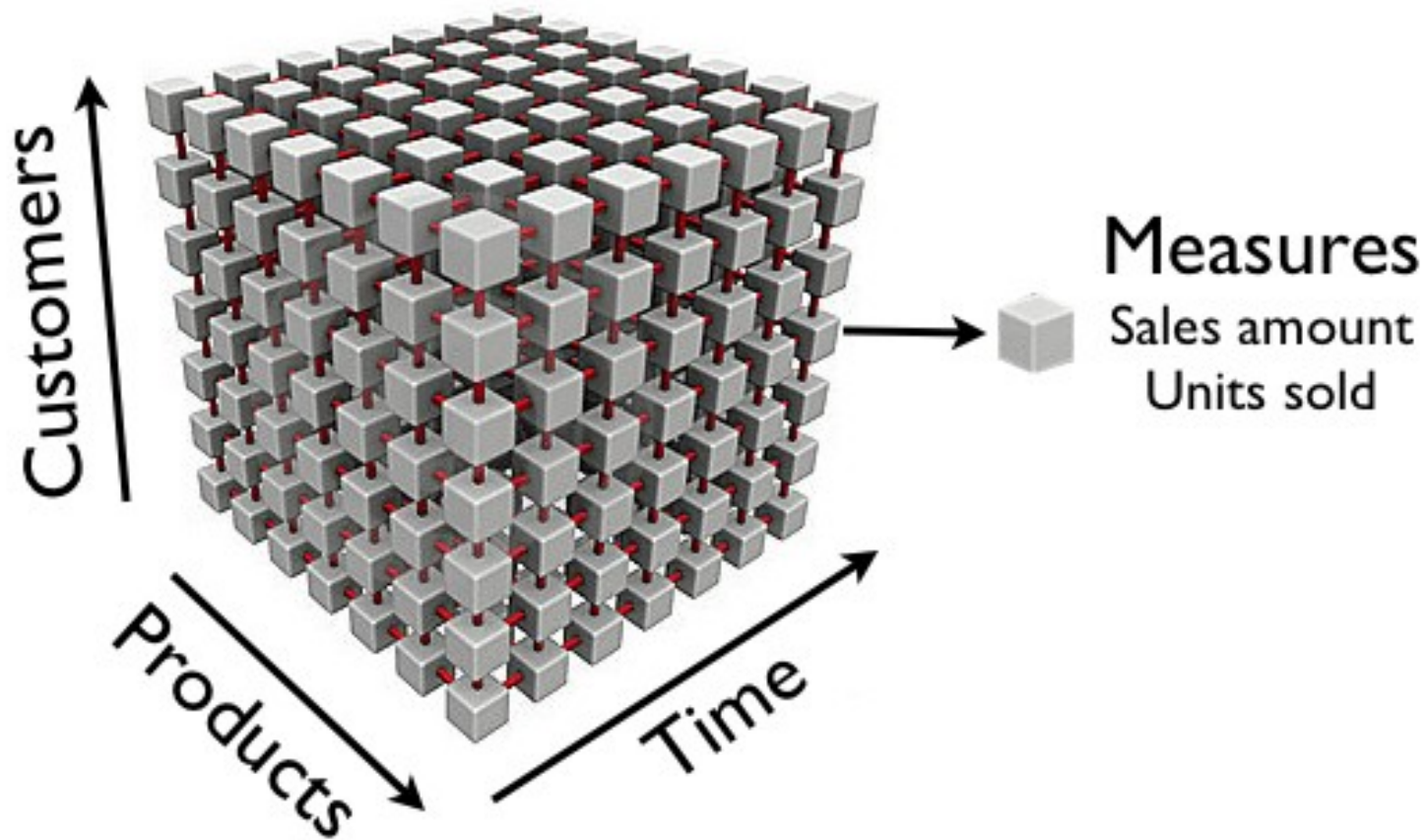
09|1957533.34

10|609445.63

11|1620398.94

12|556889.47

# Data Cube



# Build Data Cube (1)

```
select n_name as country, c_mktsegment as segment,
 substr(o_orderdate, 6, 2) as month,
 sum(o_totalprice) as tot_orders
from orders, customer, nation, region
where o_custkey = c_custkey
 and c_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and o_orderdate >= '1996-01-01'
 and o_orderdate < '1997-01-01'
 and r_name = 'AMERICA'
group by n_name, c_mktsegment, month
```

```
UNITED STATES|AUTOMOBILE|02|320234.13
UNITED STATES|AUTOMOBILE|03|41703.87
UNITED STATES|AUTOMOBILE|04|519989.12
UNITED STATES|AUTOMOBILE|05|366946.75
UNITED STATES|AUTOMOBILE|07|115110.42
UNITED STATES|AUTOMOBILE|08|133691.66
UNITED STATES|AUTOMOBILE|09|73907.33
UNITED STATES|AUTOMOBILE|11|88538.53
UNITED STATES|BUILDING|01|307934.57
UNITED STATES|BUILDING|04|449200.42
UNITED STATES|BUILDING|05|504249.66
UNITED STATES|BUILDING|06|535603.54
UNITED STATES|BUILDING|07|197825.34
```

# Build Data Cube (2)

```
select '*' as country, segment, month, tot_orders
from
```

```
 (select
 c_mktsegment as segment,
 substr(o_orderdate, 6, 2) as month,
 sum(o_totalprice) as tot_orders
 from orders, customer, nation, region
 where o_custkey = c_custkey
 and c_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and o_orderdate >= '1996-01-01'
 and o_orderdate < '1997-01-01'
 and r_name = 'AMERICA'
 group by c_mktsegment, month)
```

```
*|FURNITURE|07|1536410.0
*|FURNITURE|08|1275646.64
*|FURNITURE|09|850020.17
*|FURNITURE|10|627374.67
*|FURNITURE|11|1674244.08
*|FURNITURE|12|596075.08
*|HOUSEHOLD|01|857604.53
*|HOUSEHOLD|02|1268791.05
*|HOUSEHOLD|03|734929.51
*|HOUSEHOLD|04|977667.03
```

# Build Data Cube (3)

```
select '*' as country, '*' as segment, month, tot_orders
```

```
from
```

```
 (select
```

```
 substr(o_orderdate, 6, 2) as month,
```

```
 sum(o_totalprice) as tot_orders
```

```
from orders, customer, nation, region
```

```
where o_custkey = c_custkey
```

```
 and c_nationkey = n_nationkey
```

```
 and n_regionkey = r_regionkey
```

```
 and o_orderdate >= '1996-01-01'
```

```
 and o_orderdate < '1997-01-01'
```

```
 and r_name = 'AMERICA'
```

```
group by month)
```

```
||01|3951668.23
```

```
||02|6019772.02
```

```
||03|5905427.19
```

```
||04|5324806.52
```

```
||05|5117392.48
```

```
||06|5055926.24
```

```
||07|5821844.72
```

```
||08|7009655.79
```

```
||09|5697575.73
```

```
||10|5306528.08
```

```
||11|6371999.76
```

```
||12|4739471.8
```

# Build Data Cube (4)

```
select '*' as country, '*' as segment, '*' as month, tot_orders
from
```

```
(select
```

```
 sum(o_totalprice) as tot_orders
```

```
from orders, customer, nation, region
```

```
||*|66322068.56
```

```
where o_custkey = c_custkey
```

```
 and c_nationkey = n_nationkey
```

```
 and n_regionkey = r_regionkey
```

```
 and o_orderdate >= '1996-01-01'
```

```
 and o_orderdate < '1997-01-01'
```

```
 and r_name = 'AMERICA')
```

# SQL Data Cube Operator

**create materialized view DataCube as**

```
select n_name as country, c_mktsegment as segment,
 substr(o_orderdate, 6, 2) as month,
 sum(o_totalprice) as tot_orders
from orders, customer, nation, region
where o_custkey = c_custkey
 and c_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and o_orderdate >= '1996-01-01'
 and o_orderdate < '1997-01-01'
 and r_name = 'AMERICA'
group by n_name, c_mktsegment, month WITH CUBE
```

# Data Cube in SQLite

- create table DataCube (  
    country char(50), segment char(50), month char(10), tot\_orders decimal(20,4),  
    primary key (country, segment, month))
- insert into DataCube  
    select n\_name as country, c\_mktsegment as segment, substr(o\_orderdate, 6, 2) as month, sum(o\_totalprice) as tot\_orders  
    group by n\_name, c\_mktsegment, month
- UNION  
    select '\*' as country, segment, month, tot\_orders  
    group by c\_mktsegment, month  
    select country, '\*' as segment, month, tot\_orders  
    select country, segment, '\*' as month, tot\_orders
- UNION  
    select '\*' as country, '\*' as segment, month, tot\_orders  
    group by month  
    select '\*' as country, segment, '\*' as month, tot\_orders  
    select country, '\*' as segment, '\*' as month, tot\_orders
- UNION  
    select '\*' as country, '\*' as segment, '\*' as month, tot\_orders

# Data Exploration with Data Cube

- Data exploration with drill-down and roll-up
  - select country, tot\_orders from DataCube where segment = '\*' and month = '\*'
- Drill-down on market segment in US
  - select segment, tot\_orders from DataCube  
where country = 'UNITED STATES' and month = '\*'
- Drill-down on month for BUILDING
  - select month, tot\_orders from DataCube  
where country = 'UNITED STATES' and segment = 'BUILDING'
- Roll-up on month
  - select month, tot\_orders from DataCube  
where country = 'UNITED STATES' and segment = '\*'

# Constraints

Keys & Foreign Keys, Referential Integrity, CHECK

# Constraints

- Specify the values an attribute can take in a tuple or a table
- Specified on top of the attribute data type
- Defined in **CREATE TABLE** statement
  - [https://sqlite.org/lang\\_createtable.html](https://sqlite.org/lang_createtable.html)
  - [https://sqlite.org/lang\\_conflict.html](https://sqlite.org/lang_conflict.html)
- **Automatically checked by the database for every modification (I/U/D) operation**
  - I/U/D operations incur overhead (are slower)

# Example 1

```
CREATE TABLE Product (
 maker CHAR(32) DEFAULT('Unknown'),
 model INTEGER PRIMARY KEY,
 type VARCHAR(20) NOT NULL,
 CHECK (type IN ('pc', 'laptop', 'printer')),
 CHECK ((type = 'pc' AND model >= 1000 AND model <
2000) OR (type = 'laptop' AND model >= 2000 AND model
< 3000) OR (type = 'printer' AND model >= 3000 AND
model < 4000))
)
```

# PRIMARY KEY

- 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
- SQLite
  - Tuples are sorted on this attribute(s)
  - There is an index on PRIMARY KEY

# NOT NULL

- The value of a NOT NULL attribute cannot be NULL
- `insert into Product(model) values(1100)`
- `insert into Product(model, maker) values(1100, 'A')`
- `insert into Product(maker, type) values('A', 'pc')`
  - *model* is primary key
  - *model* is set to `MAX(model)+1` (AUTO-INCREMENT)

# DEFAULT

- When the attribute value is not specified, the DEFAULT value is used
- insert into Product(model, maker) values(1100, 'A')
  - Unknown|1100|pc

# CHECK Clause

- CHECK (type IN ('pc', 'laptop', 'printer'))
- CHECK ((type = 'pc' AND model >= 1000 AND model < 2000) OR (type = 'laptop' AND model >= 2000 AND model < 3000) OR (type = 'printer' AND model >= 3000 AND model < 4000))
  - Any valid condition that can be in the *WHERE* clause
- insert into Product values('A', 1100, 'PC')
- insert into Product values('A', 2100, 'pc')

# Example 2

```
CREATE TABLE PC (
 model INTEGER REFERENCES Product(model),
 speed FLOAT,
 ram INTEGER,
 hd INTEGER,
 price DECIMAL(7,2) NOT NULL,
 PRIMARY KEY(model)
)
```

# REFERENCES

- Foreign Key Referential Integrity
  - Cross-table attribute value constraint
  - The value of the attribute has to be one of the values of the referenced attribute (or NULL)
- SQLite
  - <https://sqlite.org/foreignkeys.html>
  - PRAGMA foreign\_keys → ON/OFF
  - The referenced attribute has to be PRIMARY KEY or UNIQUE INDEX
    - model INTEGER PRIMARY KEY
    - CREATE UNIQUE INDEX product\_idx\_model ON Product(model)
  - For efficiency, an index should be defined on the FOREIGN KEY attribute, if not already a PRIMARY KEY
    - CREATE INDEX pc\_idx\_model ON PC(model)

# Referential Integrity Operations

- model INTEGER REFERENCES Product(model)
  - **Product**
    - INSERT INTO Product VALUES('A', 1001, 'pc')
    - DELETE FROM Product WHERE model = 1001
  - **PC**
    - INSERT INTO PC(model, speed, ram, hd, price) VALUES(1001, 2.66, 1024, 250, 2114)
    - DELETE FROM PC WHERE model = 1001
- **UPDATE = DELETE + INSERT**

# DEFERRED FOREIGN KEY

- model INTEGER REFERENCES Product(model)
  - INSERT INTO PC(model, speed, ram, hd, price)  
VALUES(1001, 2.66, 1024, 250, 2114)
  - INSERT INTO Product VALUES('A', 1001, 'pc')
- model INTEGER REFERENCES Product(model)  
**DEFERRABLE INITIALLY DEFERRED**
  - BEGIN TRANSACTION
  - INSERT INTO PC(model, speed, ram, hd, price)  
VALUES(1001, 2.66, 1024, 250, 2114)
  - INSERT INTO Product VALUES('A', 1001, 'pc')
  - COMMIT TRANSACTION

# ON DELETE/UPDATE Actions

- model INTEGER REFERENCES Product(model)  
**ON DELETE CASCADE ON UPDATE SET NULL**
- Impacts operations on **Product**
- SQLite
  - NO ACTION
  - RESTRICT
  - **SET NULL**
  - SET DEFAULT
  - **CASCADE**

# Constraints Summary

- Enforce data are clean & consistent
- Database is automatically guaranteeing constraints' satisfaction
  - Overhead for modification operations (I/U/D)
    - **275 ms vs 298 ms** on small Computers database
- Modification operation (I/U/D) order becomes very important
  - Error messages are hard to understand

Triggers

# Active Databases

- The database is monitoring the modification operations (I/U/D) and (re)acts
  - Enforce **CONSTRAINTS**
    - When: for every operation
    - How: limited operations imposed by the database
  - Execute **TRIGGERS** ([https://sqlite.org/lang\\_createtrigger.html](https://sqlite.org/lang_createtrigger.html))
    - *Event-Condition-Action (ECA)* rules
    - When: only when *Event* satisfies *Condition*
    - How: fully customizable programmer *Action*

# Example 1

CREATE TRIGGER insertPC\_no\_exists **BEFORE INSERT** ON PC

**FOR EACH ROW**

**EVENT:** I/U/D operation

**WHEN** (NOT EXISTS (

**BEFORE** or **AFTER**

select \*

**CONDITION:** anything that goes in WHERE

from Product p, PriceRange pr

where p.model = **NEW.model**

and p.maker = pr.maker

and p.type = pr.type))

**I/U/D operation is performed  
independent of the trigger execution**

**FOR EACH ROW**

**FOR EACH STATEMENT:** **not in SQLite**

**BEGIN**

insert into PriceRange

**ACTION:** I/U/D operation(s)

select maker, type, **NEW.price**, **NEW.price**

from Product

where model = **NEW.model**;

**NEW:** the tuple that gets inserted

**END**

# Example 2

```
CREATE TRIGGER deleteLaptop_all AFTER DELETE ON Laptop
FOR EACH ROW
WHEN (OLD.price = (select maxPrice from Product p, PriceRange pr
 where p.model = OLD.model and p.maker = pr.maker and p.type = pr.type)
AND OLD.price = (select minPrice from Product p, PriceRange pr
 where p.model = OLD.model and p.maker = pr.maker and p.type = pr.type))
BEGIN
 delete from PriceRange
 where maker = (select maker
 from Product p
 where p.model = OLD.model)
 and type = (select type
 from Product p
 where p.model = OLD.model);
END
```

**OLD:** the tuple that got deleted

# Example 3

```
CREATE TRIGGER updatePrinter_min AFTER UPDATE ON Printer
FOR EACH ROW
```

```
WHEN (NEW.price < (select minPrice from Product p, PriceRange pr
 where p.model = NEW.model and p.maker = pr.maker and p.type = pr.type))
```

```
BEGIN
```

```
 update PriceRange
```

```
 set minPrice = NEW.price
```

```
 where maker = (select maker
```

```
 from Product p
```

```
 where p.model = NEW.model)
```

```
 and type = (select type
```

```
 from Product p
```

```
 where p.model = NEW.model);
```

```
END
```

**OLD:** the old value of the tuple that got updated

**NEW:** the new value of the tuple that got updated

# Example 4

CREATE TRIGGER insertPC\_Maker **INSTEAD OF INSERT** ON  
**PC\_Maker**

**FOR EACH ROW**

- Trigger is executed instead of I/U/D operation
- Allows for I/U/D operations on views

**BEGIN**

insert into Product(model, maker, type)  
values(NEW.model, NEW.maker, 'pc');

insert into PC

values(NEW.model, NEW.speed, NEW.ram, NEW.hd, NEW.price);

**END**

# Triggers Summary

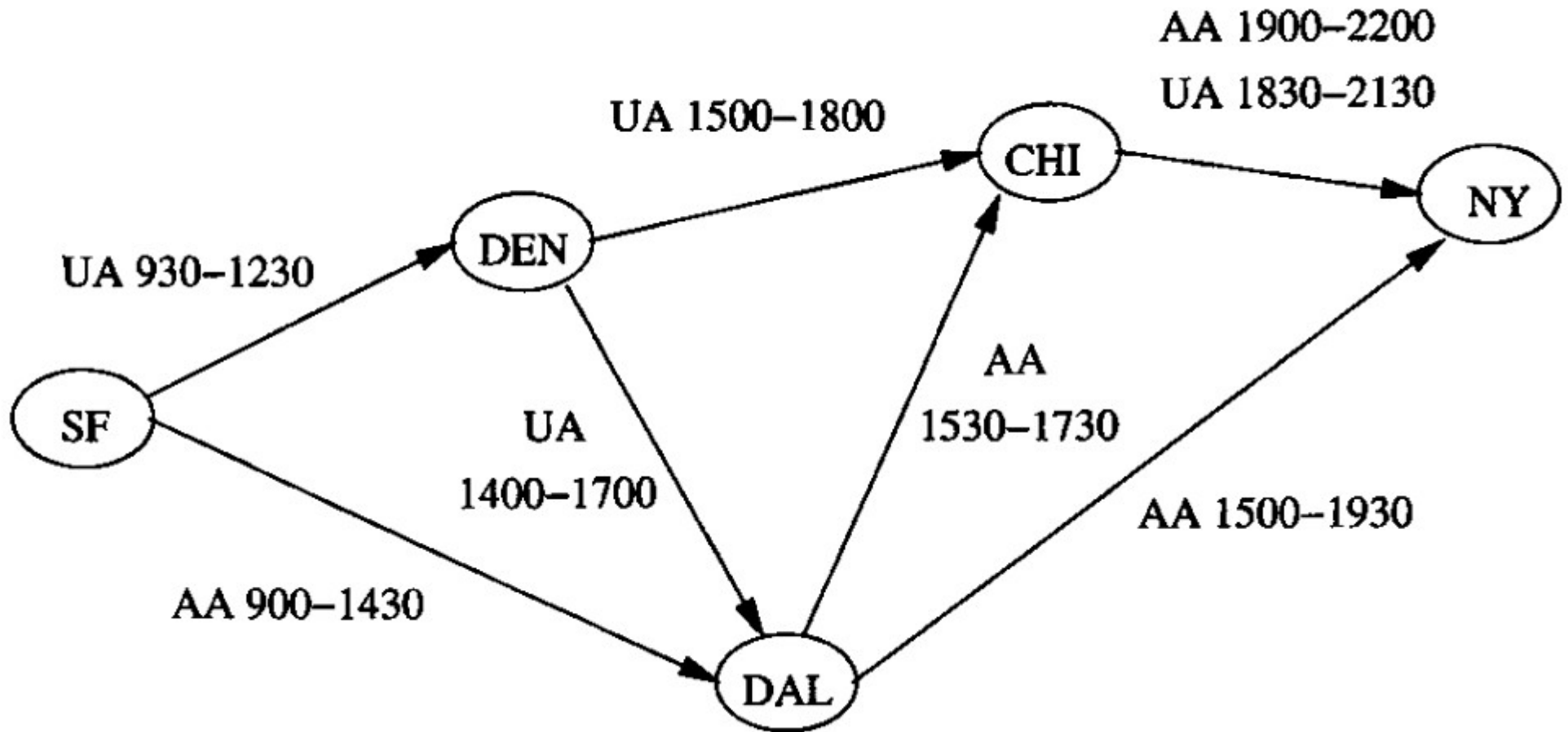
- Programmer has complete control
  - How constraints are enforced
  - How modification operations (I/U/D) are handled
- Unexpected interaction with I/U/D operations
- Transform views into base tables
- Implement materialized view maintenance

# Recursion in SQL

# SQL WITH Clause

- create view PC\_Maker(model, speed, ram, hd, price, maker) as  
select PC.model, speed, ram, hd, price, maker  
from PC, Product P  
where PC.model = P.model
- **with** **PC\_Maker**(model, speed, ram, hd, price, maker) as  
(select PC.model, speed, ram, hd, price, maker  
from PC, Product P  
where PC.model = P.model)  
select \* from **PC\_Maker**
- *[https://sqlite.org/lang\\_with.html](https://sqlite.org/lang_with.html)*

# Graph Reachability



# Graph Reachability with Recursion

- `Flights(orig, dest, depart, arrive)`
- `Reaches(orig, dest)`
  - `Reaches(o,d) <= Flights(o,d)`
    - base case
  - `Reaches(o,d) <= Reaches(o,x) AND Flights(x,d)`
    - recursive case
  - `Reaches(o,d) <= Reaches(o,x) AND Reaches(x,d)`

# SQL Recursion

- **with recursive** **Reaches**(orig, dest) as

(select orig, dest

from Flights

**base case**

**union**

select r.orig, f.dest

from **Reaches** r, Flights f

**recursive case**

where r.dest = f.orig)

select \* from **Reaches**

# Graph Reachability with Constraints

- Reaches(orig, dest, depart, arrive)
  - $\text{Reaches}(o,d, \text{dep}, \text{arr}) \leq \text{Flights}(o,d, \text{dep}, \text{arr})$ 
    - base case
  - $\text{Reaches}(o,d, d1, a2) \leq \text{Reaches}(o,x, d1, a1) \text{ AND } \text{Flights}(x,d, d2, a2) \text{ AND } d2-a1 > 100$ 
    - recursive case

# SQL Query

- with recursive Reaches(orig, dest, depart, arrive) as  
    (select orig, dest, depart, arrive  
    from Flights  
    union  
    select r.orig, f.dest, r.depart, f.arrive  
    from Reaches r, Flights f  
    where r.dest = f.orig  
        and f.depart-r.arrive > 100)  
select \* from Reaches

# Median PC Price with LIMIT

- ```
SELECT AVG(price)
FROM (
  SELECT price
  FROM PC
  ORDER BY price
  LIMIT 2 - (SELECT COUNT(*) FROM PC) % 2  -- odd 1, even 2
  OFFSET (SELECT (COUNT(*) - 1) / 2 FROM PC)
)
```
- *<https://stackoverflow.com/questions/15763965/how-can-i-calculate-the-median-of-values-in-sqlite>*

Median PC Price with Ranking

- **with recursive**
- G(model_1, model_2, diff) as
(select p1.model, p2.model, p2.price -
p1.price
from PC p1, PC p2
where p1.price <= p2.price),
- **Hops**(model_1, model_2, hop) as
(select model_1, model_2, 0
from G
union
select h.model_1, G.model_2, h.hop+1
from **Hops** h, G
where h.model_2 = G.model_1
and G.diff > 0),
- Rank(model, rnk) as
(select model_2, max(hop)
+1 as rank
from Hops h
group by model_2
order by rank)
- select *
from Rank
where rnk = (select count(*)/2
from Rank)