CS 295B/CS 395B Systems for Knowledge Discovery

Lecture 4:

Example presentations

The University *of* Vermont

Guidelines

- Presentations should be 18-20minutes
- Rough rating system
  - Bad – Factually incorrect or no discernable content
  - Fair – Regurgitates facts from the paper in the same order
  - Good – Provides narrative cohesion and insights beyond surface text of paper
  - Excellent – Advertises and entertains while teaching the audience something
- Tip: do extra reading for context, read related or cited work, etc.

You cannot cover everything; make editorial choices

Try to mix things up, visually.

Outline

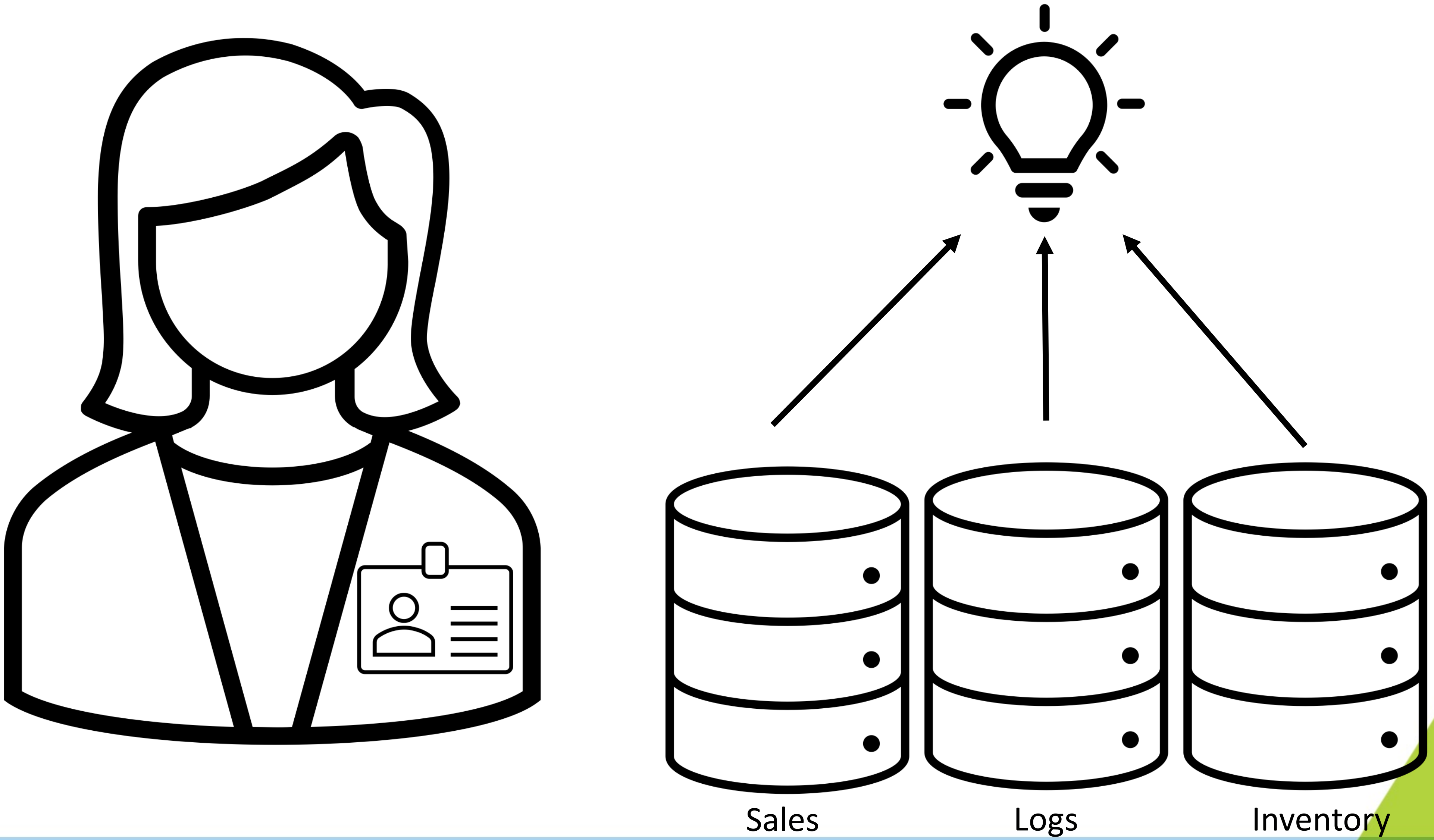
- Overview
- Example presentation 1 (excellent)
- Example presentation 2 (fair to good)
- Analysis of presentations

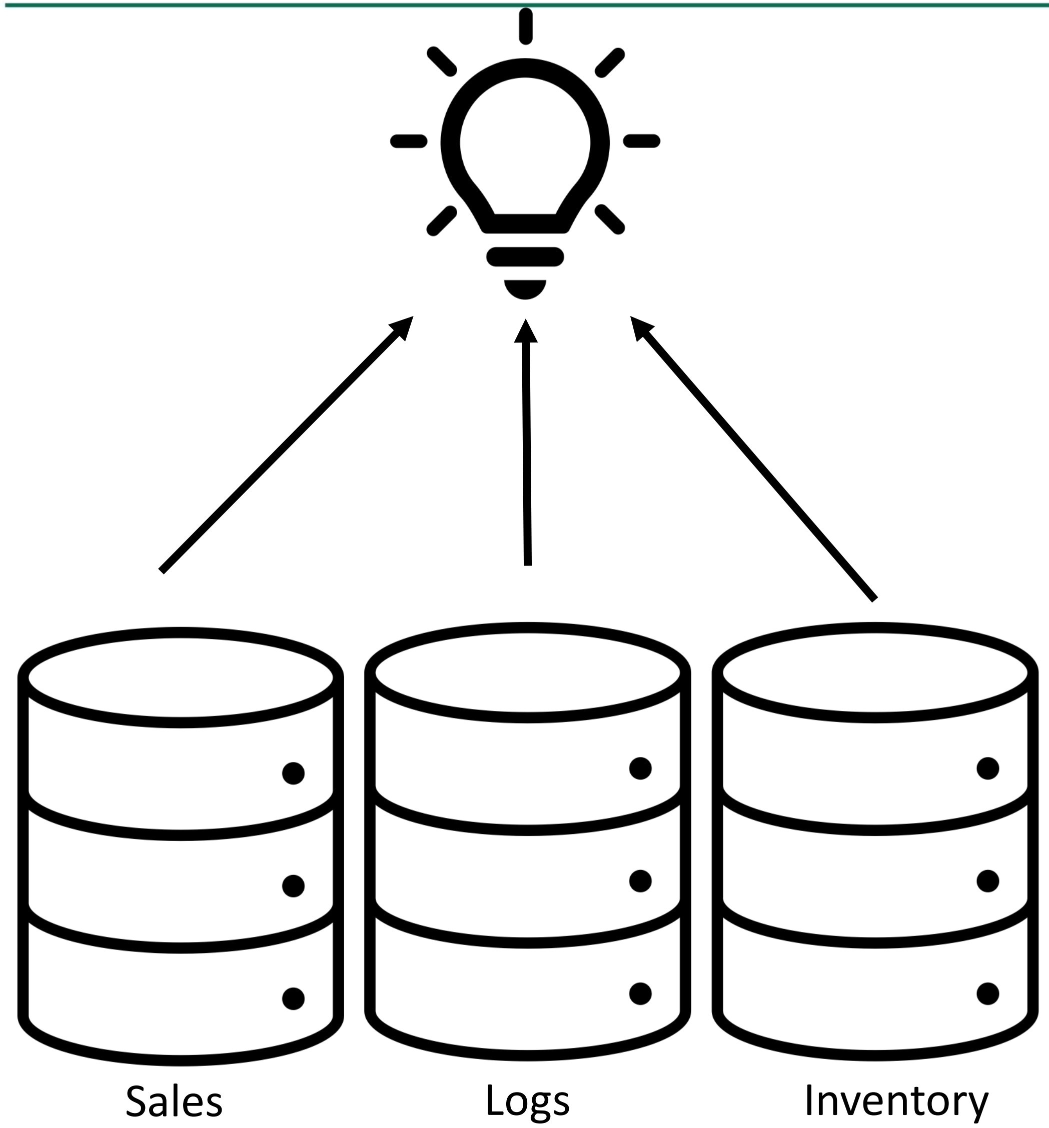
Systems for KDD: From Concepts to Practice

Authors: Dunkel et al.

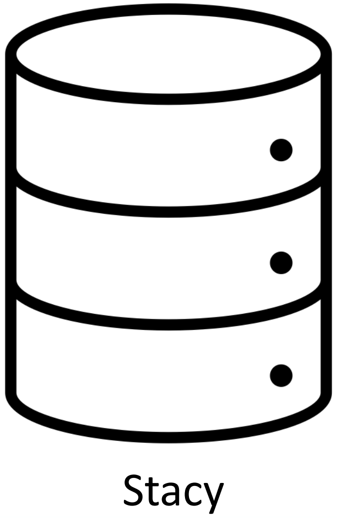
Presenter: Emma Tosch

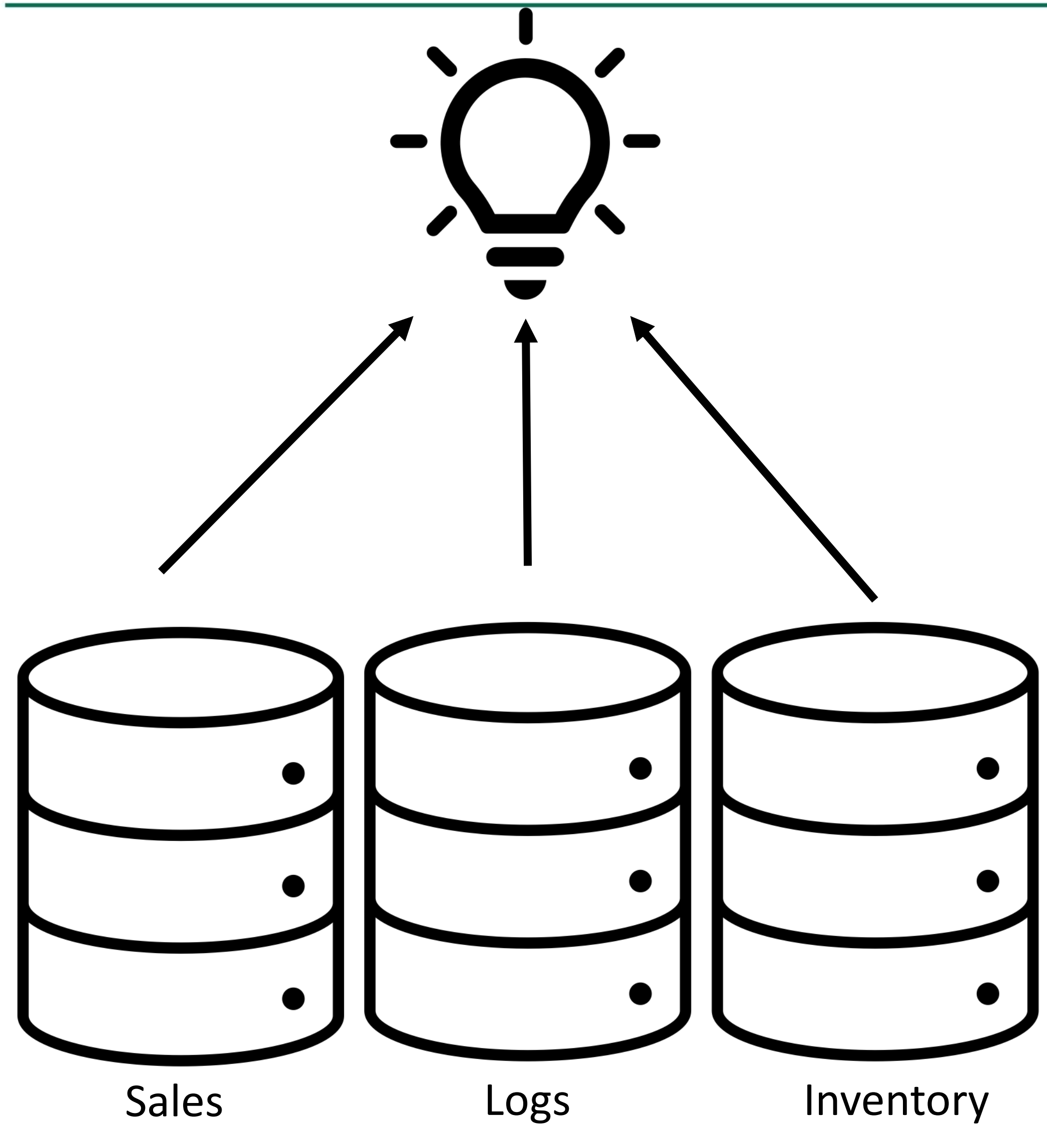
The University of Vermont


Sales
Logs
Inventory

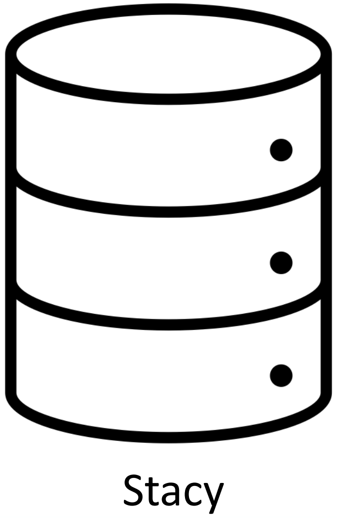


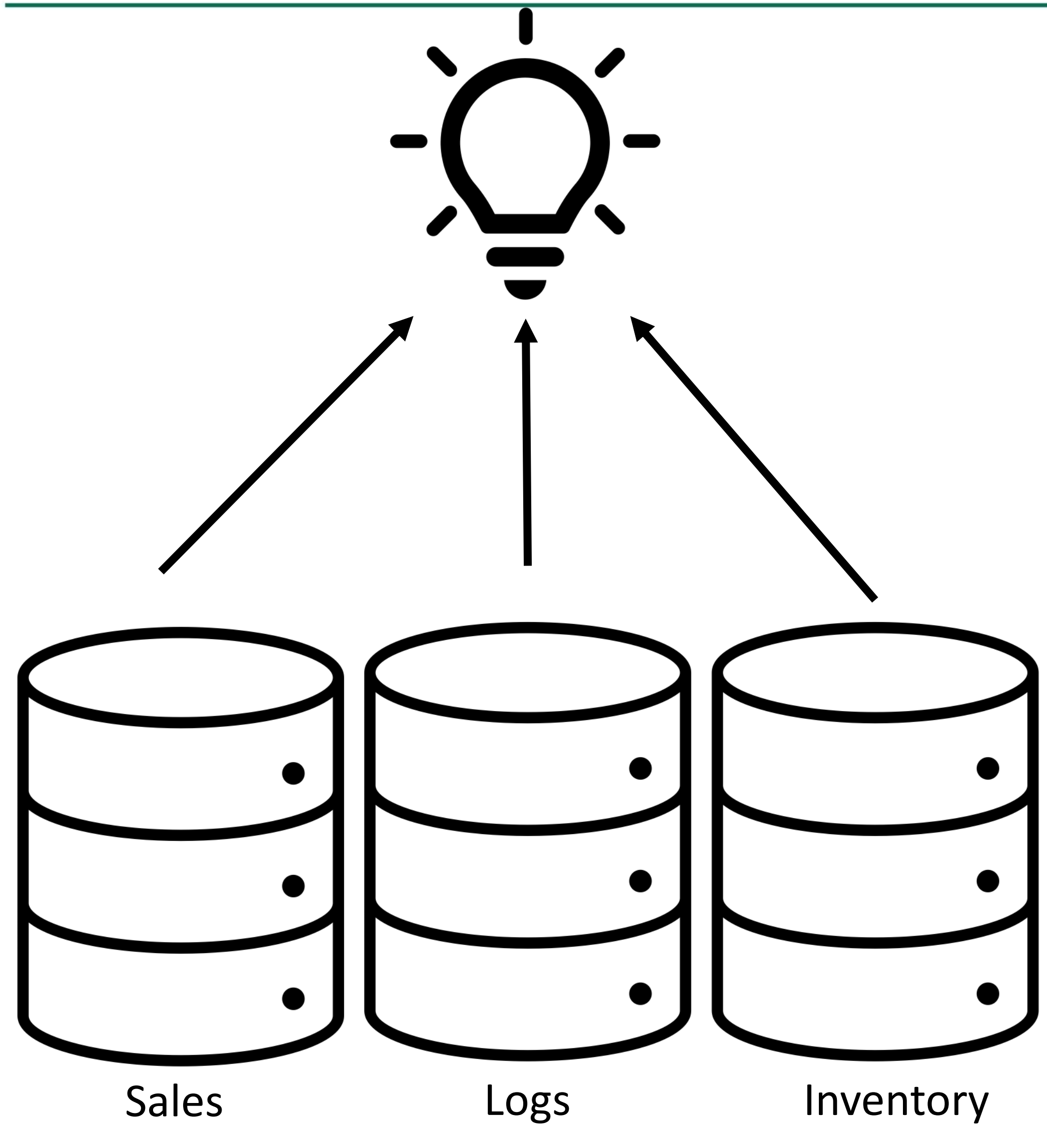
```
SELECT    S.date, S.item, S.tid, S.iid
          L.clicks, L.time, L.region,
          I.stock
INTO      Stacy
FROM      Sales as S, Logs as L, Inventory as I
WHERE     S.tid = L.tid
AND       S.iid = I.iid
```



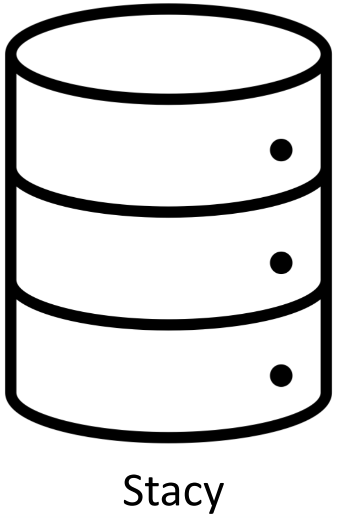


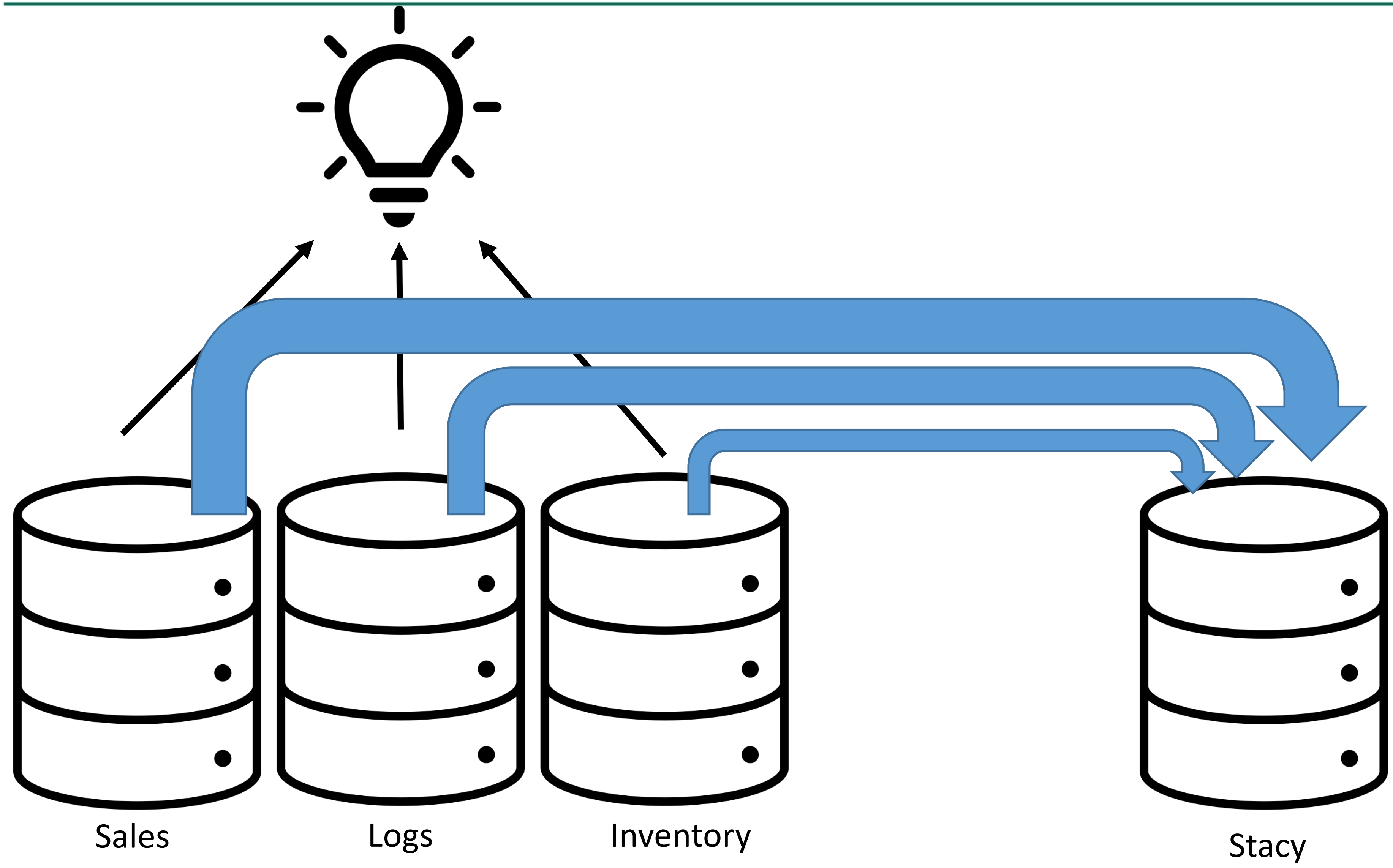
```
SELECT   AVG(time), AVG(clicks)
FROM     Stacy
GROUP BY tid
```

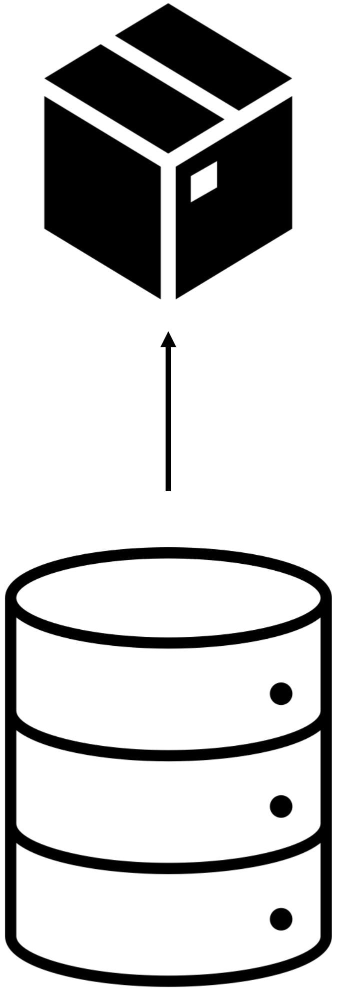


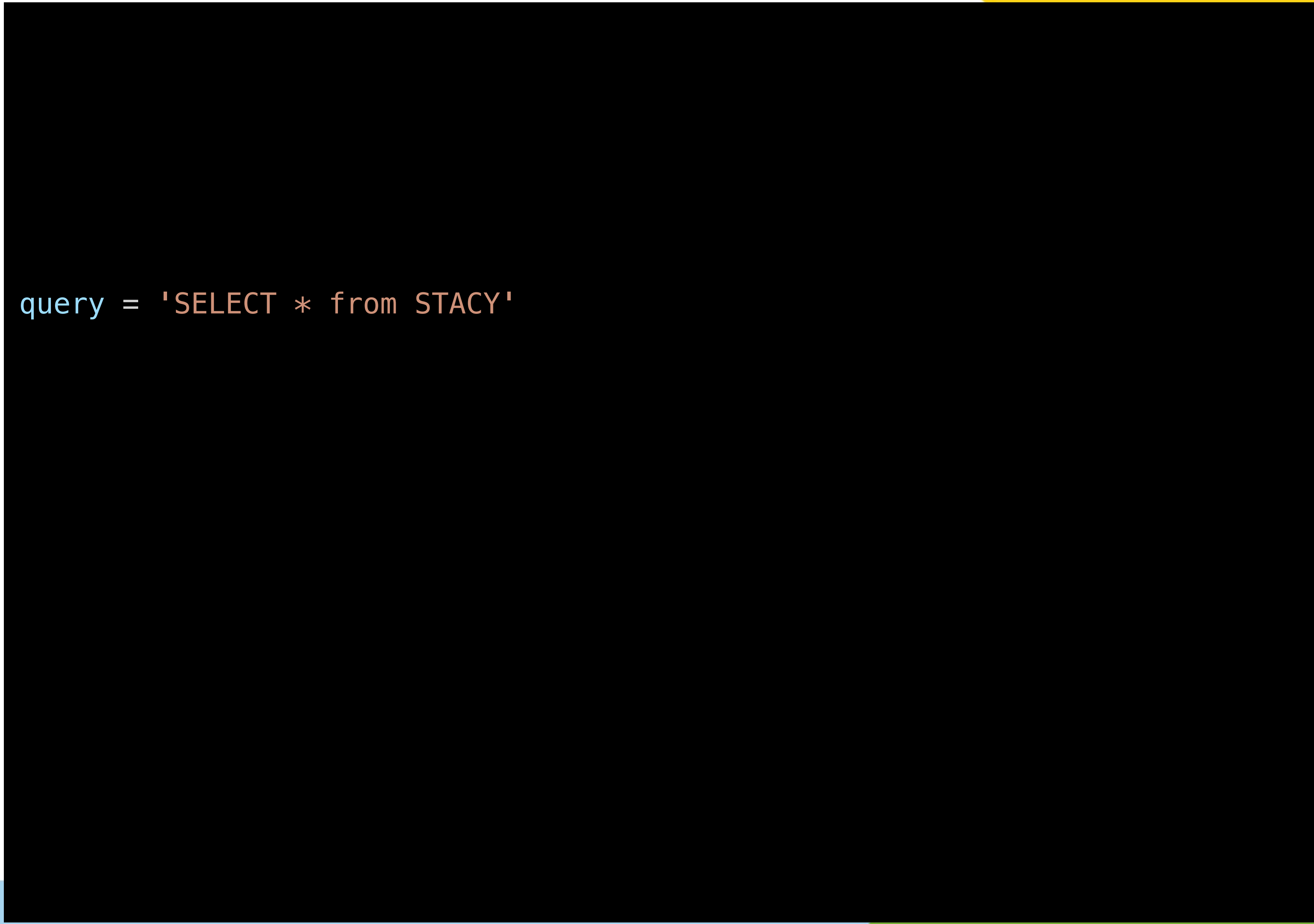


```
SELECT    *
FROM      Stacy
WHERE     region is NULL
```

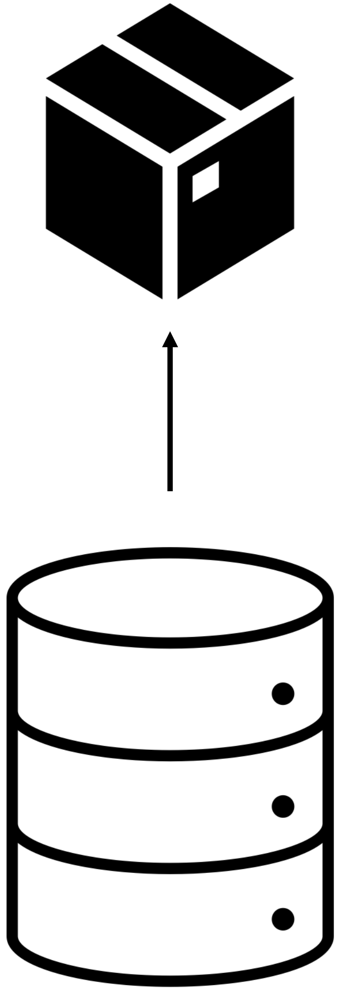



Sales
Logs
Inventory
Stacy



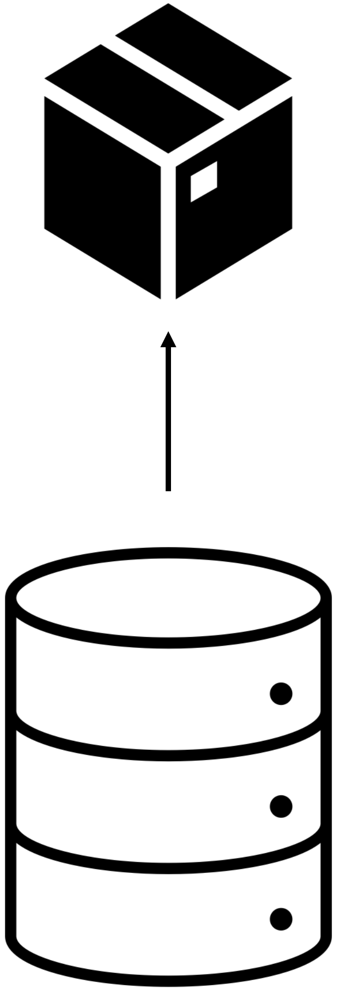


```
query = 'SELECT * from STACY'
```

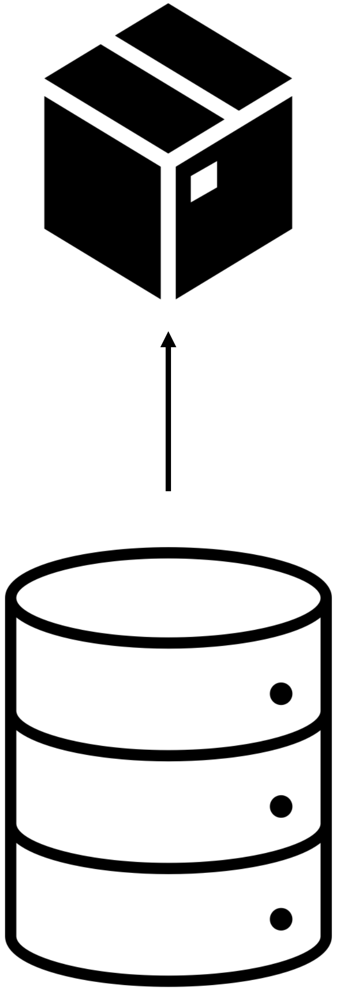


Stacy

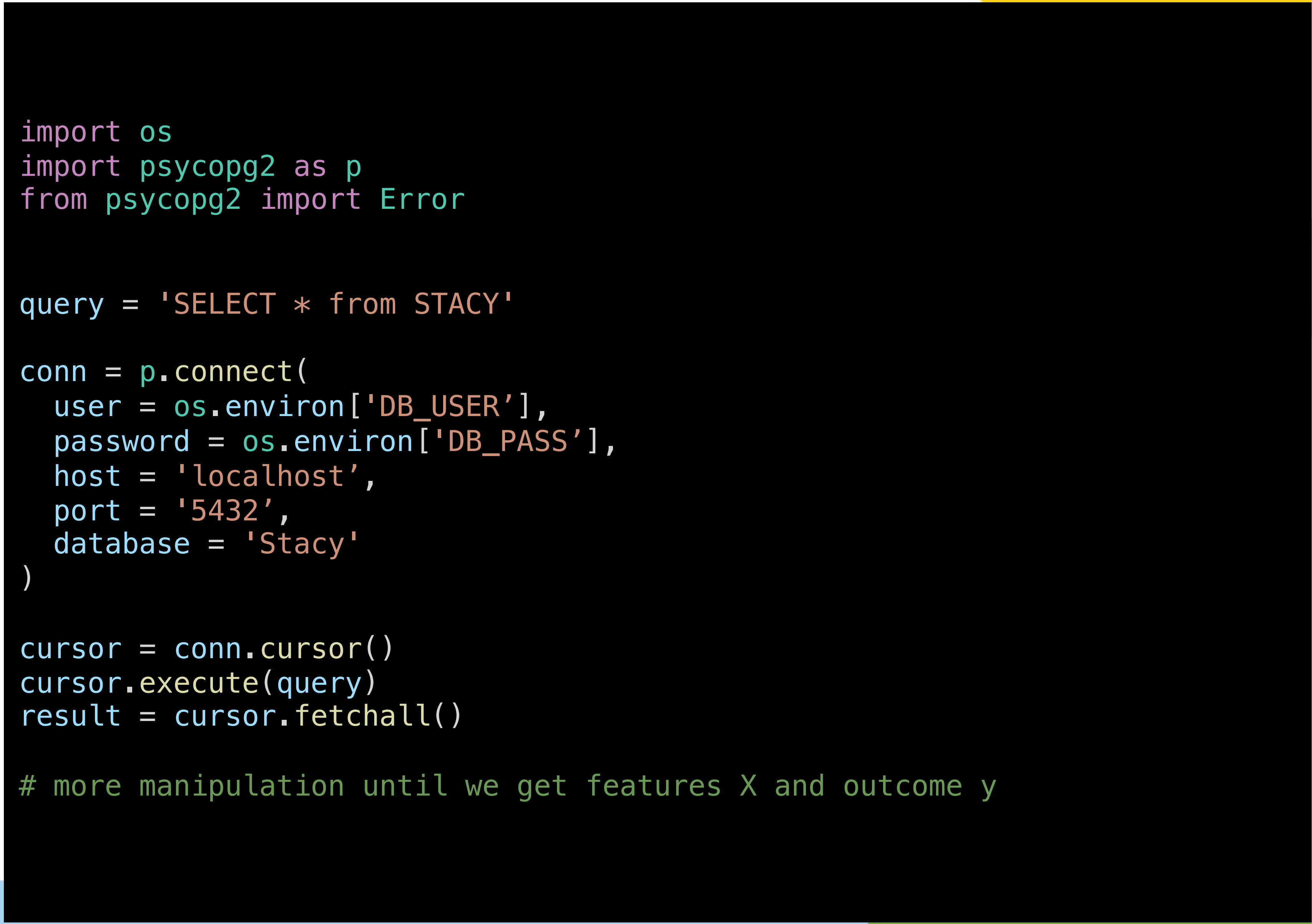
```
import os
import psycopg2 as p
from psycopg2 import Error


query = 'SELECT * from STACY'

conn = p.connect(
  user = os.environ['DB_USER’],
  password = os.environ['DB_PASS’],
  host = 'localhost’,
  port = '5432’,
  database = 'Stacy'
)
```



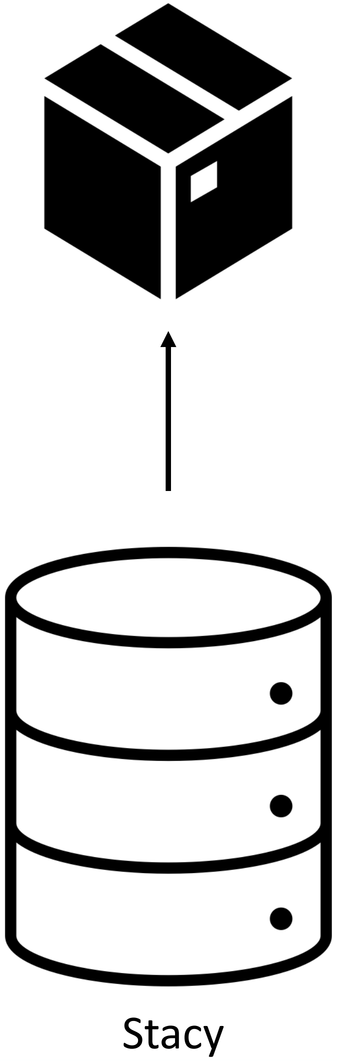
```
import os
import psycopg2 as p
from psycopg2 import Error


query = 'SELECT * from STACY'

conn = p.connect(
  user = os.environ['DB_USER’],
  password = os.environ['DB_PASS’],
  host = 'localhost’,
  port = '5432’,
  database = 'Stacy'
)

cursor = conn.cursor()
cursor.execute(query)
result = cursor.fetchall()
```

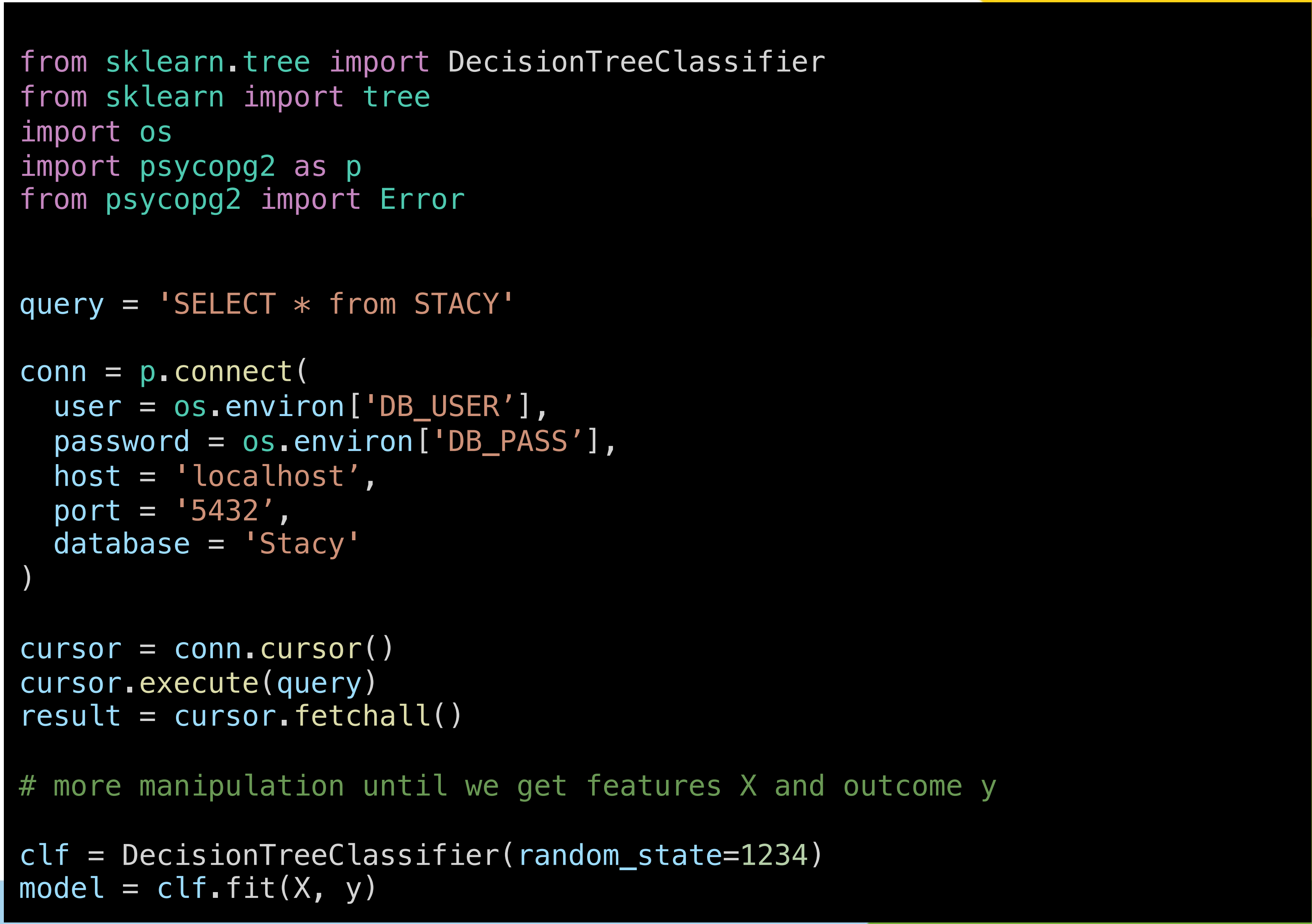


Stacy

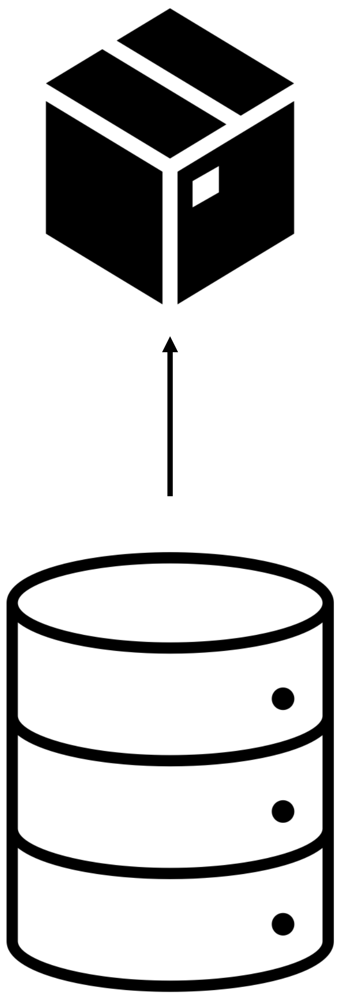


```
import os
import psycopg2 as p
from psycopg2 import Error


query = 'SELECT * from STACY'

conn = p.connect(
  user = os.environ['DB_USER’],
  password = os.environ['DB_PASS’],
  host = 'localhost’,
  port = '5432’,
  database = 'Stacy'
)

cursor = conn.cursor()
cursor.execute(query)
result = cursor.fetchall()

# more manipulation until we get features X and outcome y
```



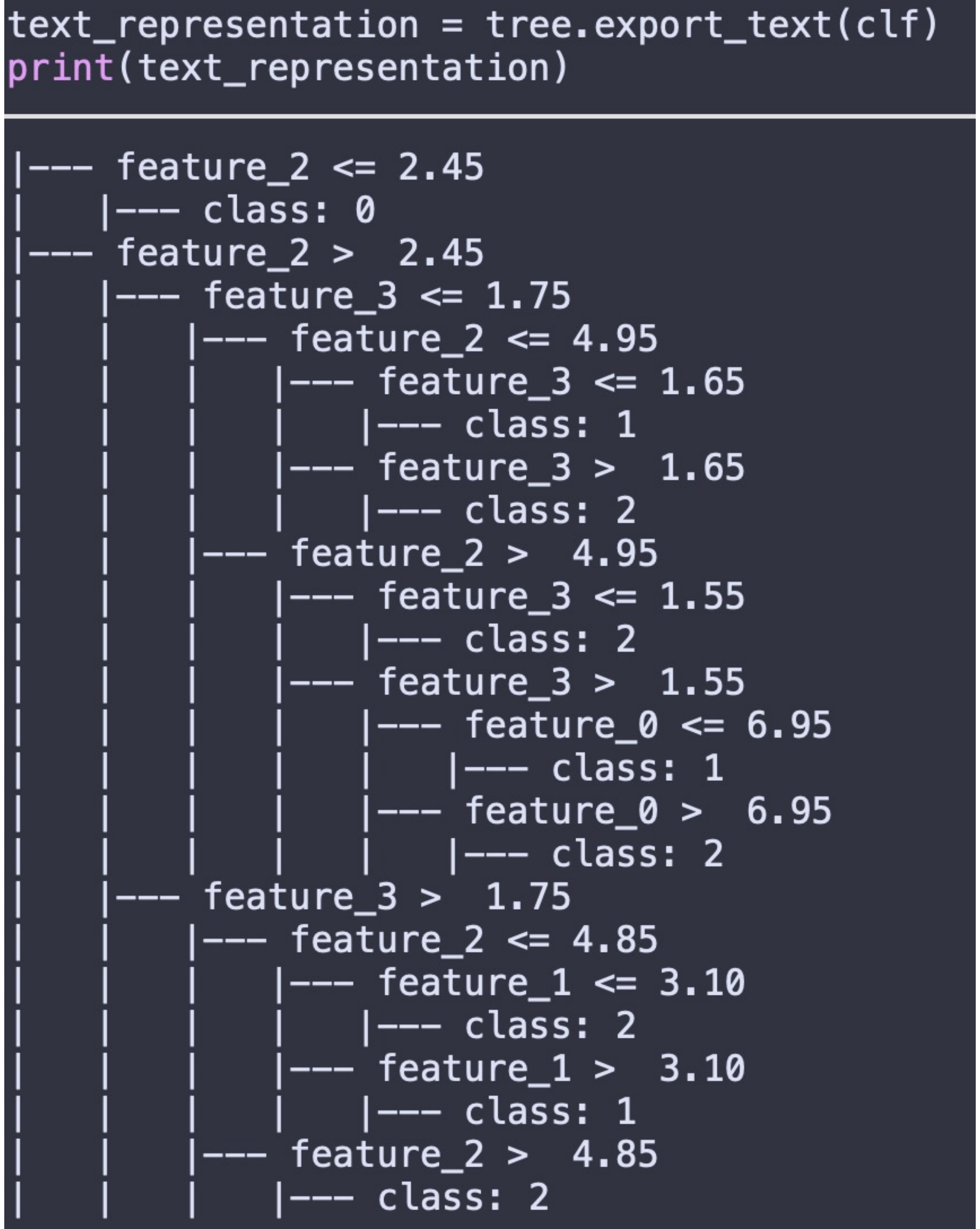


```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import os
import psycopg2 as p
from psycopg2 import Error


query = 'SELECT * from STACY'

conn = p.connect(
  user = os.environ['DB_USER’],
  password = os.environ['DB_PASS’],
  host = 'localhost’,
  port = '5432’,
  database = 'Stacy'
)

cursor = conn.cursor()
cursor.execute(query)
result = cursor.fetchall()

# more manipulation until we get features X and outcome y

clf = DecisionTreeClassifier(random_state=1234)
model = clf.fit(X, y)
```



Stacy

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import os
import psycopg2 as p
from psycopg2 import Error


query = 'SELECT * from STACY'

conn = p.connect(
  user = os.environ['DB_USER’],
  password = os.environ['DB_PASS’],
  host = 'localhost’,
  port = '5432’,
  database = 'Stacy'
)

cursor = conn.cursor()
cursor.execute(query)
result = cursor.fetchall()

# more manipulation until we get features X and outcome y

clf = DecisionTreeClassifier(random_state=1234)
model = clf.fit(X, y)
```



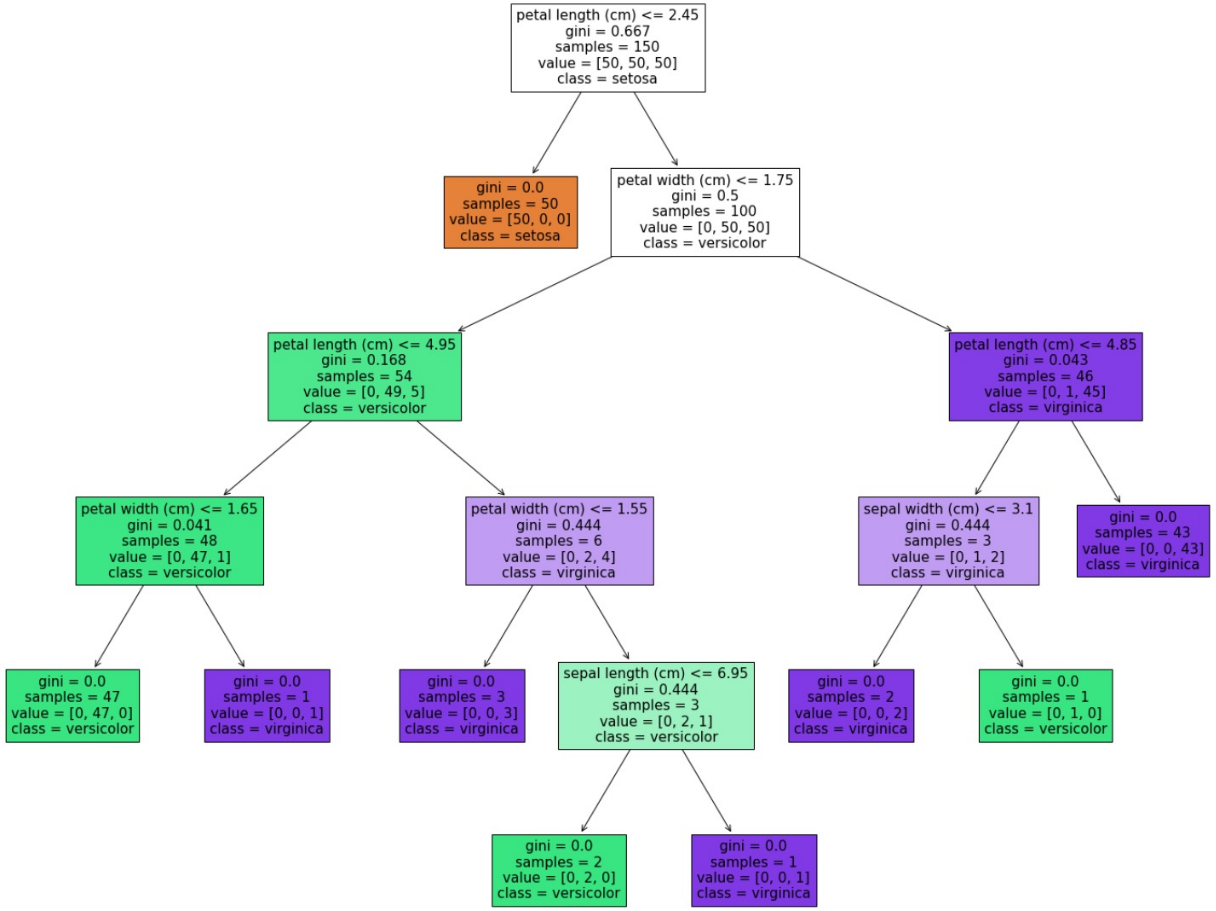
```
text_representation = tree.export_text(clf)
print(text_representation)
```

```
|--- feature_2 <= 2.45
|   |--- class: 0
|--- feature_2 >  2.45
|   |--- feature_3 <= 1.75
|   |   |--- feature_2 <= 4.95
|   |   |   |--- feature_3 <= 1.65
|   |   |   |   |--- class: 1
|   |   |   |--- feature_3 >  1.65
|   |   |   |   |--- class: 2
|   |   |--- feature_2 >  4.95
|   |   |   |--- feature_3 <= 1.55
|   |   |   |   |--- class: 2
|   |   |   |--- feature_3 >  1.55
|   |   |   |   |--- feature_0 <= 6.95
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_0 >  6.95
|   |   |   |   |   |--- class: 2
|   |--- feature_3 >  1.75
|   |   |--- feature_2 <= 4.85
|   |   |   |--- feature_1 <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- feature_1 >  3.10
|   |   |   |   |--- class: 1
|   |   |--- feature_2 >  4.85
|   |   |   |--- class: 2
```

A quick look at the output

- Interpret as a bunch of if-statements
- Remember: the output is a class (e.g., binary classifier for sale of item class)
- Can be hard to read

https://mljar.com/blog/visualize-decision-tree/

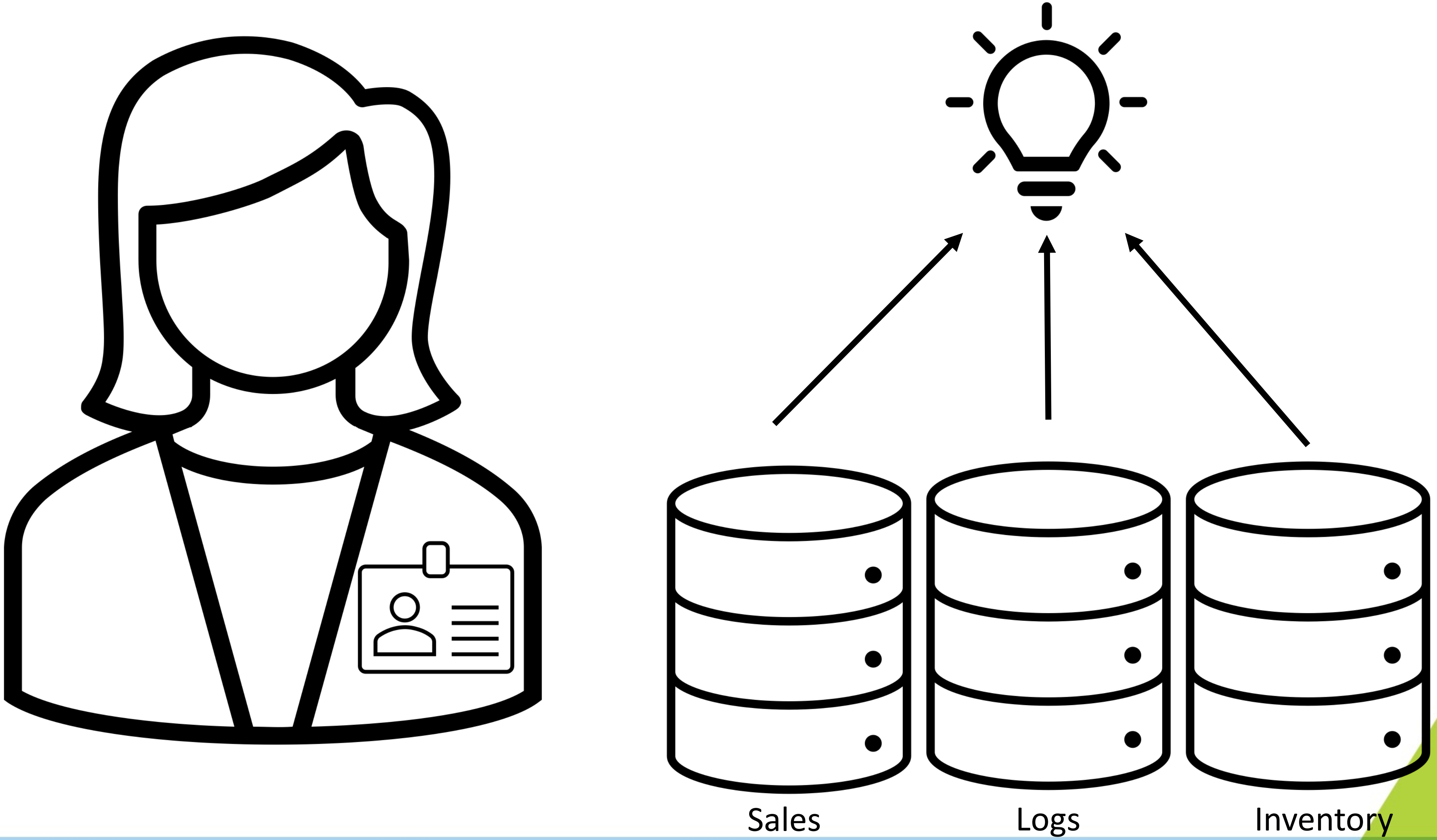


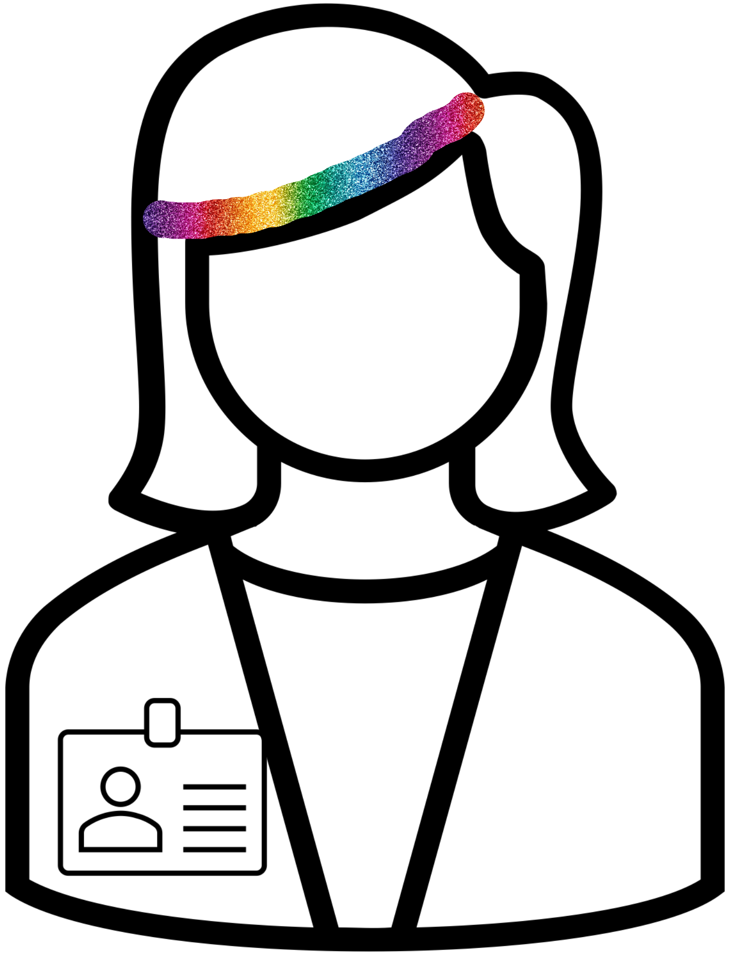
https://mljar.com/blog/visualize-decision-tree/

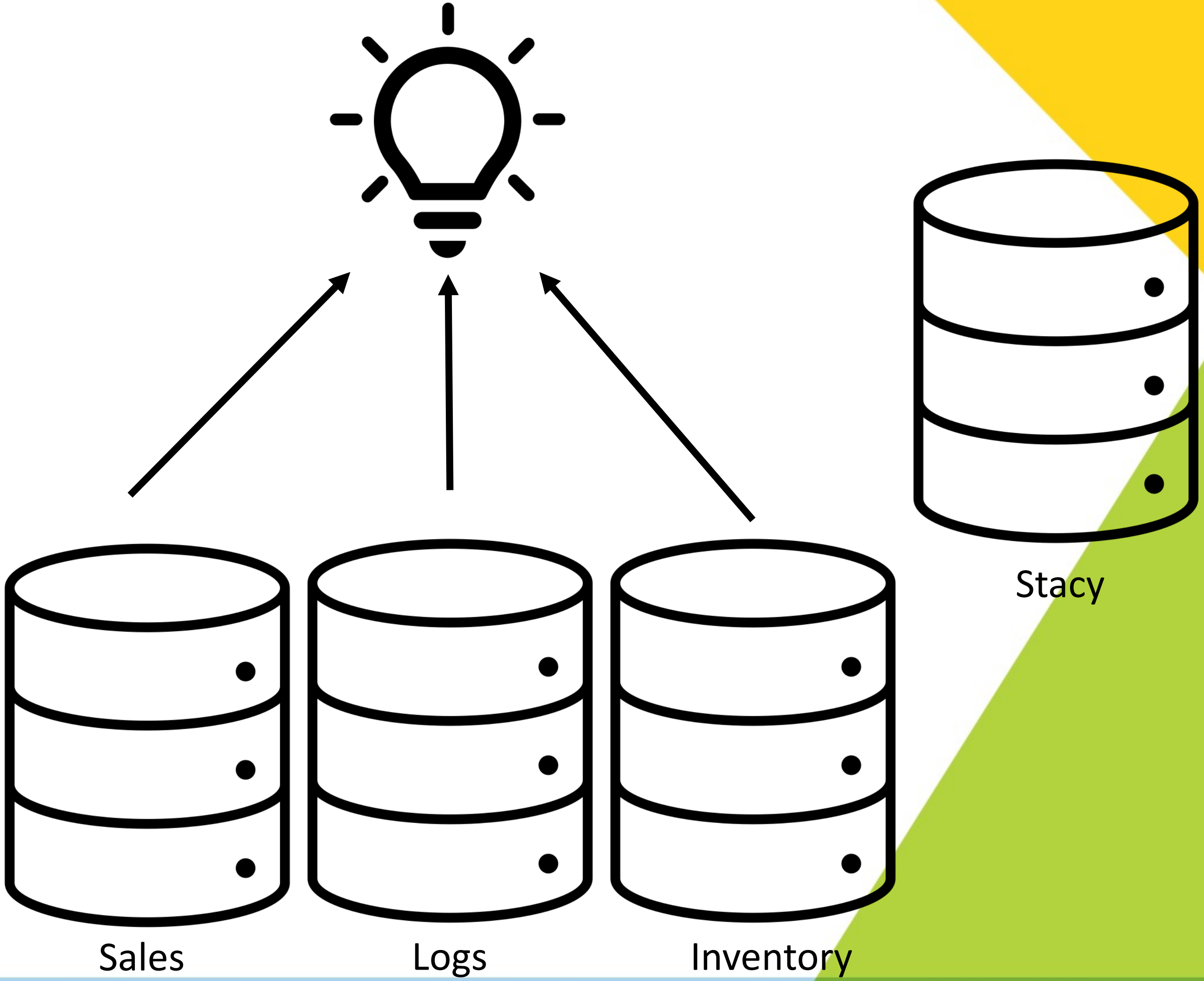
Visualize!

- Load up new library (here, matplotlib)
- Use colors to indicate majority class at this node
- Can be shown to non-domain-experts

What could go wrong?


Sales
Logs
Inventory



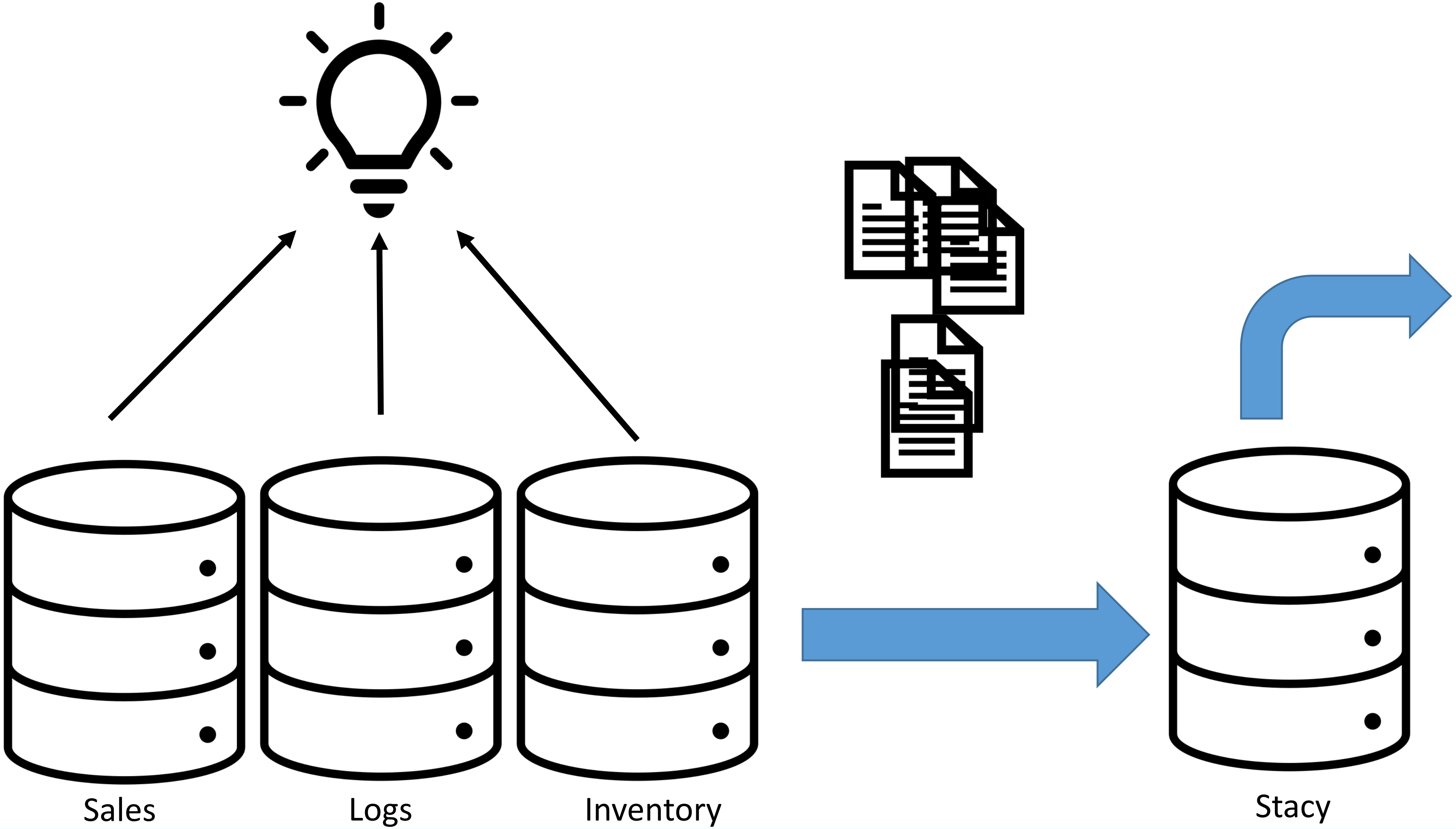


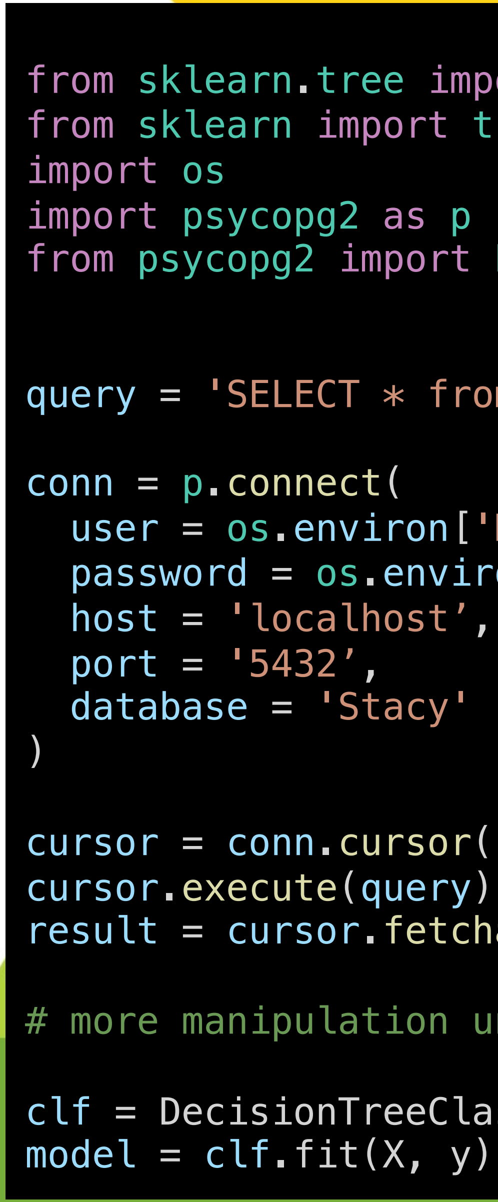
Sales

Logs

Inventory

Stacy





```
from sklearn.tree imp
from sklearn import t
import os
import psycopg2 as p
from psycopg2 import 


query = 'SELECT * fro

conn = p.connect(
  user = os.environ['
  password = os.envir
  host = 'localhost’,
  port = '5432’,
  database = 'Stacy'
)

cursor = conn.cursor(
cursor.execute(query)
result = cursor.fetch

# more manipulation u

clf = DecisionTreeCla
model = clf.fit(X, y)
```

Tightly couple the steps of the KDD process…

Here, use a KDD-specific tool!

```
from sklearn.tree imp
from sklearn import t
import os
                    s p
from psycopg2 import

query = 'SELECT * fro

conn = p.connect(
  user = os.environ['
  password = os.envir
  host = 'localhost',
  port = '5432',
  database = 'Stacy'

          conn.cursor(
      .execute(query)
result = cursor.fetch

# more manipulation


clf = DecisionTreeCla
model = clf.fit(X, y)
```

Empirical Study: Approach

Goal Idea: build a system using principles from existing systems

Identify desirable components, prototype idealized system

Overview: existing system support for KDD

System	Select	Clean	Transform	Mine	Interpret	Evaluate
Intelligent Miner		✓	✓			
MineSet		✓	✓		✓	✓
MLC++				✓	✓	
Clementine	✓	✓		✓	✓	
DBMiner (includes GeoMiner)				✓		
IDIS	✓	✓				
Mobal	✓	✓				
DataSurveyor						
Emerald						✓

Overview: existing system support for KDD

System	Select	Clean	Transform	Mine	Interpret	Evaluate
Intelligent Miner		✓	✓			
MineSet		✓	✓		✓	✓
MLC++				✓	✓	
Clementine	✓	✓		✓	✓	
DBMiner (includes GeoMiner)				✓		
IDIS	✓	✓				
Mobal	✓	✓				
DataSurveyor						
Emerald						✓

Overview: existing system support for KDD

System	Select	Clean	Transform	Mine	Interpret	Evaluate
Intelligent Miner		✓	✓			
MineSet		✓	✓		✓	✓
MLC++				✓	✓	
Clementine	✓	✓		✓	✓	
DBMiner (includes GeoMiner)				✓		
IDIS	✓	✓				
Mobal	✓	✓				
DataSurveyor						
Emerald						✓

Overview: existing system support for KDD

System	Select	Clean	Transform	Mine	Interpret	Evaluate
Intelligent Miner		✓	✓			
MineSet		✓	✓		✓	✓
MLC++				✓	✓	
Clementine	✓	✓		✓	✓	
DBMiner (includes GeoMiner)				✓		
IDIS	✓	✓				
Mobal	✓	✓				
DataSurveyor						
Emerald						✓

Overview: existing system support for KDD

System	Select	Clean	Transform	Mine	Interpret	Evaluate
Intelligent Miner		✓	✓			
MineSet		✓	✓		✓	✓
MLC++				✓	✓	
Clementine	✓	✓		✓	✓	
DBMiner (includes GeoMiner)				✓		
IDIS	✓	✓				
Mobal	✓	✓				
DataSurveyor						
Emerald						✓

Overview: existing system support for KDD

System	Select	Clean	Transform	Mine	Interpret	Evaluate
Intelligent Miner		✓	✓			
MineSet		✓	✓		✓	✓
MLC++				✓	✓	
Clementine	✓	✓		✓	✓	
DBMiner (includes GeoMiner)				✓		
IDIS	✓	✓				
Mobal	✓	✓				
DataSurveyor						
Emerald						✓

Existing system support for DM

System	Neural Nets	Rule Induction	Decision Trees	Other	Generalization	Characterization	Association
Intelligent Miner							
MineSet		✓	✓				
MLC++			✓				
Clementine	✓	✓					
DBMiner (includes GeoMiner)				✓	✓	✓	✓
IDIS							
Mobal							
DataSurveyor							
Emerald							

Existing system support for DM

System	Neural Nets	Rule Induction	Decision Trees	Other	Generalization	Characterization	Association
Intelligent Miner							
MineSet		✓	✓				
MLC++			✓				
Clementine	✓	✓					
DBMiner (includes GeoMiner)				✓	✓	✓	✓
IDIS							
Mobal							
DataSurveyor							
Emerald							

Existing systems: systems considerations

System	Het. HW/OS	Parallelization /Efficiency	Modularity	API/DX Interop	Code Generation	Easy Iteration
Intelligent Miner	✗			✓		
MineSet	✗		✓	✓		
MLC++				✓		
Clementine	✓				✓	✓
DBMiner (includes GeoMiner)	✓					✓
IDIS	✓	✓				
Mobal				✓		✓
DataSurveyor		✓				
Emerald			✓			✓

Existing systems: systems considerations

System	Het. HW/OS	Parallelization /Efficiency	Modularity	API/DX Interop	Code Generation	Easy Iteration
Intelligent Miner	✗			✓		
MineSet	✗		✓	✓		
MLC++				✓		
Clementine	✓				✓	✓
DBMiner (includes GeoMiner)	✓					✓
IDIS	✓	✓				
Mobal				✓		✓
DataSurveyor		✓				
Emerald			✓			✓

Existing systems: systems considerations

System	Het. HW/OS	Parallelization /Efficiency	Modularity	API/DX Interop	Code Generation	Easy Iteration
Intelligent Miner	✗			✓		
MineSet	✗		✓	✓		
MLC++				✓		
Clementine	✓				✓	✓
DBMiner (includes GeoMiner)	✓					✓
IDIS	✓	✓				
Mobal				✓		✓
DataSurveyor		✓				
Emerald			✓			✓

Existing systems: systems considerations

System	Het. HW/OS	Parallelization /Efficiency	Modularity	API/DX Interop	Code Generation	Easy Iteration
Intelligent Miner	✗			✓		
MineSet	✗		✓	✓		
MLC++				✓		
Clementine	✓				✓	✓
DBMiner (includes GeoMiner)	✓					✓
IDIS	✓	✓				
Mobal				✓		✓
DataSurveyor		✓				
Emerald			✓			✓

Existing systems: systems considerations

System	Het. HW/OS	Parallelization /Efficiency	Modularity	API/DX Interop	Code Generation	Easy Iteration
Intelligent Miner	✗			✓		
MineSet	✗		✓	✓		
MLC++				✓		
Clementine	✓				✓	✓
DBMiner (includes GeoMiner)	✓					✓
IDIS	✓	✓				
Mobal				✓		✓
DataSurveyor		✓				
Emerald			✓			✓

Existing systems: systems considerations

System	Het. HW/OS	Parallelization /Efficiency	Modularity	API/DX Interop	Code Generation	Easy Iteration
Intelligent Miner	✗			✓		
MineSet	✗		✓	✓		
MLC++				✓		
Clementine	✓				✓	✓
DBMiner (includes GeoMiner)	✓					✓
IDIS	✓	✓				
Mobal				✓		✓
DataSurveyor		✓				
Emerald			✓			✓

Conclusions

- Building KDD systems is hard, but worthwhile
- Existing systems (as of the 90s) supported a wide array of functionality
- Iterative human-centered parts are the hardest

A Database Perspective on Knowledge Discovery

Authors: Imielinksi and Mannila

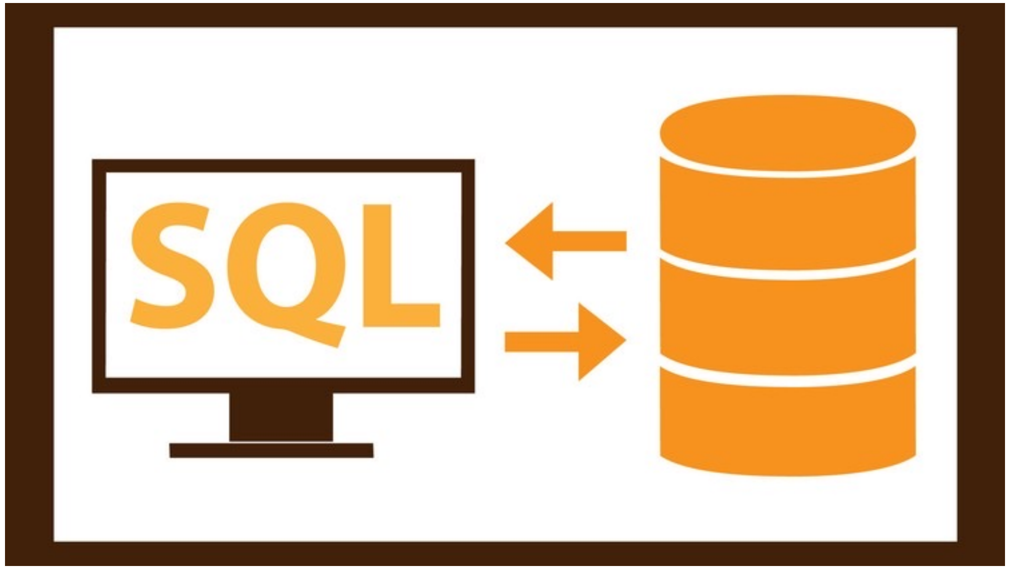
Presenter: Emma Tosch

The University of Vermont

Context

- Not a research article *per se*
- Appeared in the Communications of the ACM (CACM) in November 1996
- For a general CS audience

What is SQL?



[This Photo](#) by Unknown Author is licensed under [CC BY-NC](#)

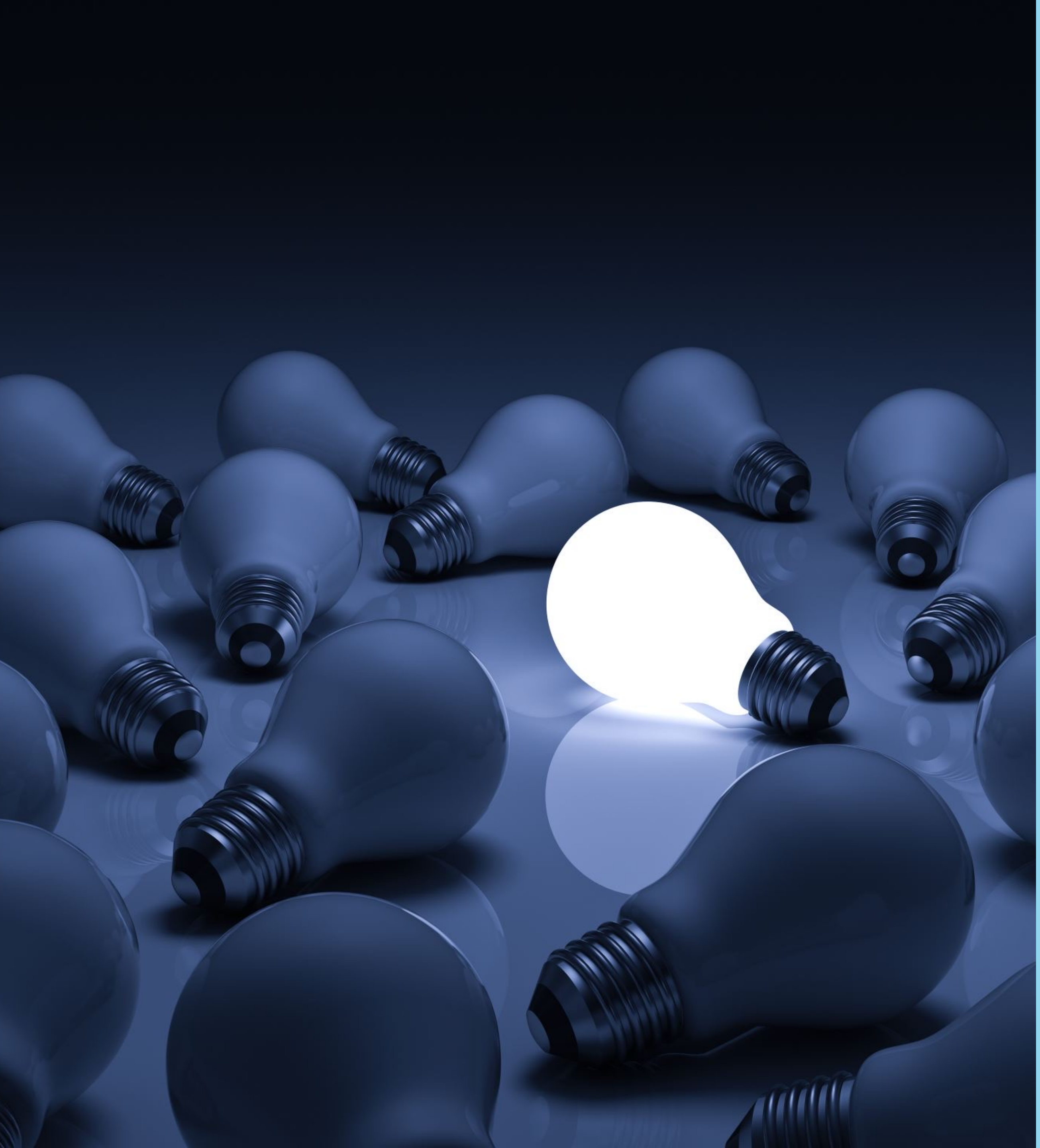
- ”Structured Query Language”
- Small number of primitives
  - SELECT columns from tables
  - Filter data WHERE constraints are true
  - JOIN tables on columns (i.e., find rows that match content)
  - Compute aggregates (AVG, COUNT, VAR)
- Returns data (as tuples) from a database

What are *ad hoc* queries?

SELECT <col+> FROM <tab+> INNER JOIN <constraint*>

(not the complete grammar)

Combinatorial grammar == endless possibilities
(can’t optimize for specific tables or joins)



Big Questions

- Performance: How do we make the KDD process faster?
- Functionality: How do we do closure?

This Photo by Unknown Author is licensed under CC BY-NC

Performance

- Article doesn’t go into much detail
- Tighter coupling
- The future will push for this
  - Analogy with I/O and traditional database systems

Functionality

- Question: What do we want knowledge querying systems to be able to do?
- Answer: all the things, very well
- How? Closure.

What is the problem with existing KDD systems (in 1996)?

- Not pluggable
- Specific to a particular data mining technique
- Basically no re-usable components
- Data mining disconnected (conceptually) from data storage
- What to do with "KDD objects?"

Closure

- Want to compose queries and KDD objects
- Queries can be regular SQL queries or special KDD SQL queries
- Closure allows embedding in a host language or application

Why rule generation is Hard

Consider simple association rules (Horn clauses):

P1, P2, .., Pn → Q

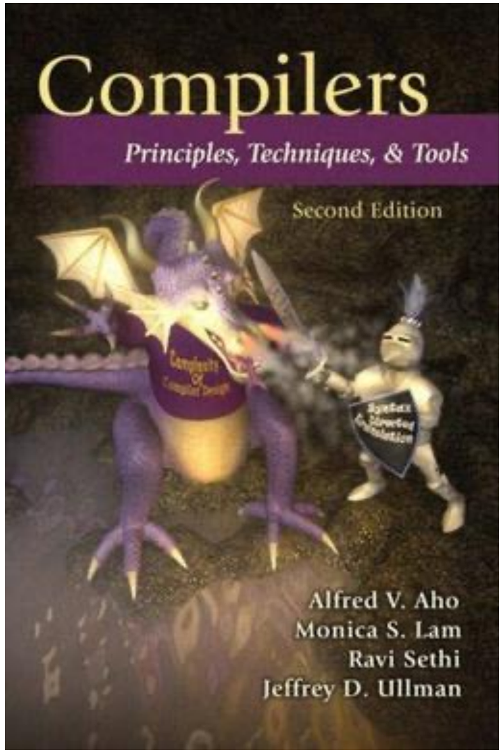
Total number of possible rules is exponential in the number of columns *in the simplest case*.

(enumerating these is what we mean by "*Any database implicitly defines the collection of all propositional or predicate rules in it.*")

All rules share structure

- Body: P1, ... Pn
- Consequent: Q
- Support: number of data points (used to compute power?)
- Confidence: Frequency
- Rules look like querying!

Good knowledge queries can be compiled



This Photo by Unknown Author is licensed under CC BY-SA

- “regular” SQL is compiled and optimized
- Need support for high-level primitives and composition
- Clever optimizations come from the data itself

Conclusion

This Photo by Unknown Author is licensed under CC BY-SA

Querying tools must support KDD objects for interoperability and optimized performance!

Meta

The University of Vermont

Talk structure

- Ground the work with an example, story, or context
- Identify the problem and why it is important
- State the solution
- Walk the audience through the high-level components of the solution
- Focus on easy-to-understand examples (remember: the talk is an advertisement)

Good: summarizing factual information

Closure

- Want to compose queries and KDD objects
- Queries can be regular SQL queries or special KDD SQL queries
- Closure allows embedding in a host language or application

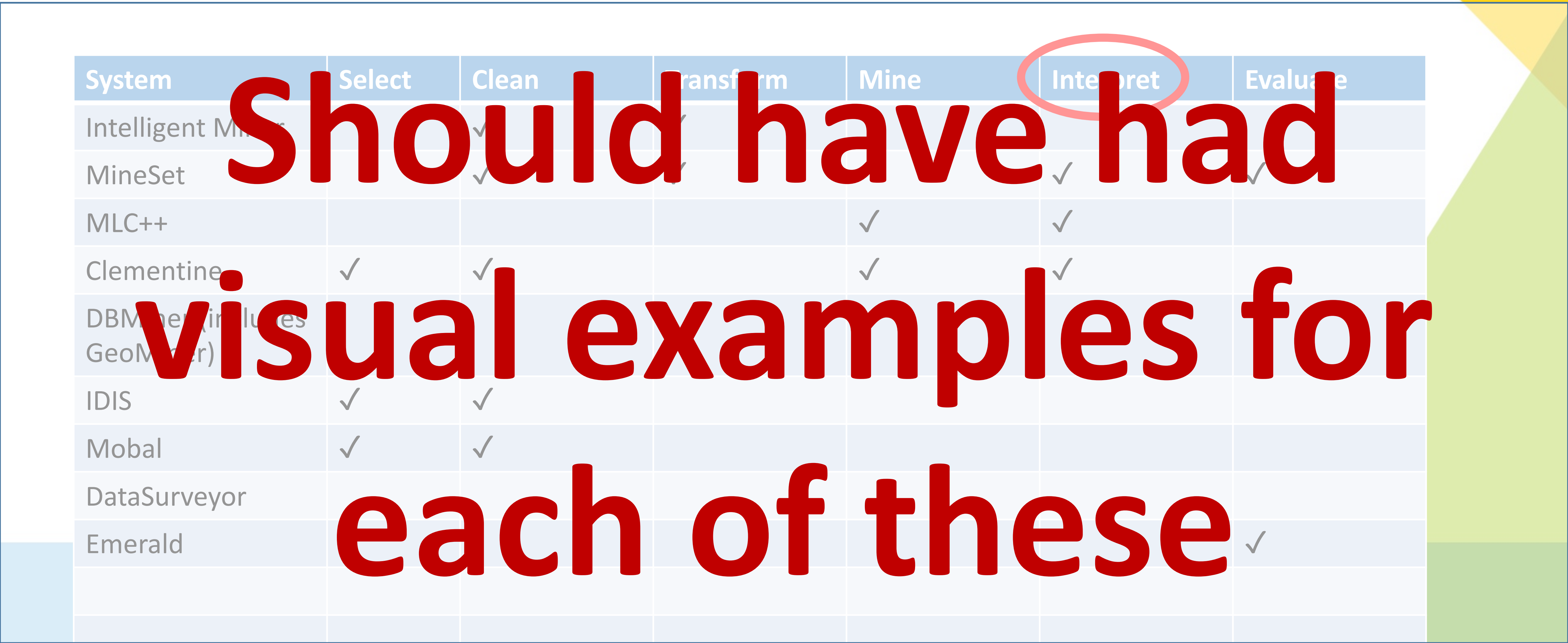
Excellent: *synthesizing* factual information into an easily-digestable format

Existing systems: systems considerations

System	Het. HW/OS	Parallelization /Efficiency	Modularity	API/DX Interop	Code Generation	Easy Iteration
Intelligent Miner	✗			✓		
MineSet	✗		✓	✓		
MLC++				✓		
Clementine	✓				✓	✓
DBMiner (includes GeoMiner)	✓					✓
IDIS	✓	✓				
Mobal				✓		✓
DataSurveyor		✓				
Emerald			✓			✓

An excellent presentation can always be improved

Overview: existing system support for KDD



System	Select	Clean	Transform	Mine	Interpret	Evaluate
Intelligent Miner		✓	✓			
MineSet		✓	✓		✓	✓
MLC++				✓	✓	
Clementine	✓	✓		✓	✓	
DBMiner (includes GeoMiner)						
IDIS	✓	✓				
Mobal	✓	✓				
DataSurveyor						
Emerald						✓

Should have had visual examples for each of these

Good luck!