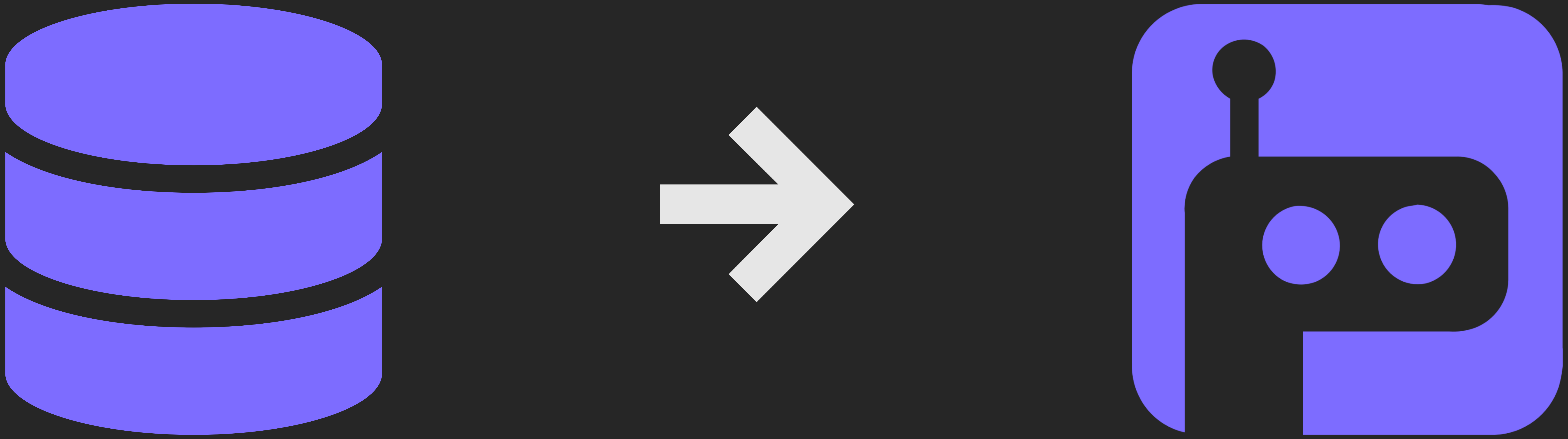


SQL -> Graql

Created by: Brandon Fergerson

Agenda

- **Setup Grakn**
- **What is a graph?**
- **Why use graphs?**
- **Are hypergraphs useful?**
- **How do I model a graph?**
- **Introducing Grakn**
  - Ontology & Knowledge Model
  - Hierarchies & Relations
  - Modeling Tips
- **Introducing Graql**
  - Defining Schemas
  - Writing Data
  - Reading Data
  - Deleting/Modifying Data
- **Exercises**

Setup Grakn

Setup Grakn & Grakn Workbase

- Download grakn-core-all-* from https://github.com/graknlabs/grakn/releases/tag/1.6.2
- Extract grakn-core-all, open terminal/CMD, and CD into extracted directory
- Run `./grakn server start` (or `.\grakn.bat server start` on Windows)

- Download grakn-workbase-* from https://github.com/graknlabs/workbase/releases/tag/1.2.7
- Extract grakn-workbase, open terminal, and CD into extracted directory
  - On Windows, double-click the .exe to install
- Run `./grakn-workbase`
  - On Windows, run installed Grakn Workbase app

Download/Clone Training Data

https://github.com/BFergerson/sql-to-graql

Executing Graql Files

- On Linux/mac:
  - ./grakn console -f /path/to/file.gql

- On Windows:
  - .\grakn.bat console -f C:\path\to\file.gql

Notice:

- Files under "graql" are standalone and can be executed separately or together in any order
- Files under "answers/exercise" are standalone and can be executed separately or together in any order
- Files under "answers/fill-in-the-blank" must be executed in the order they appear in this presentation

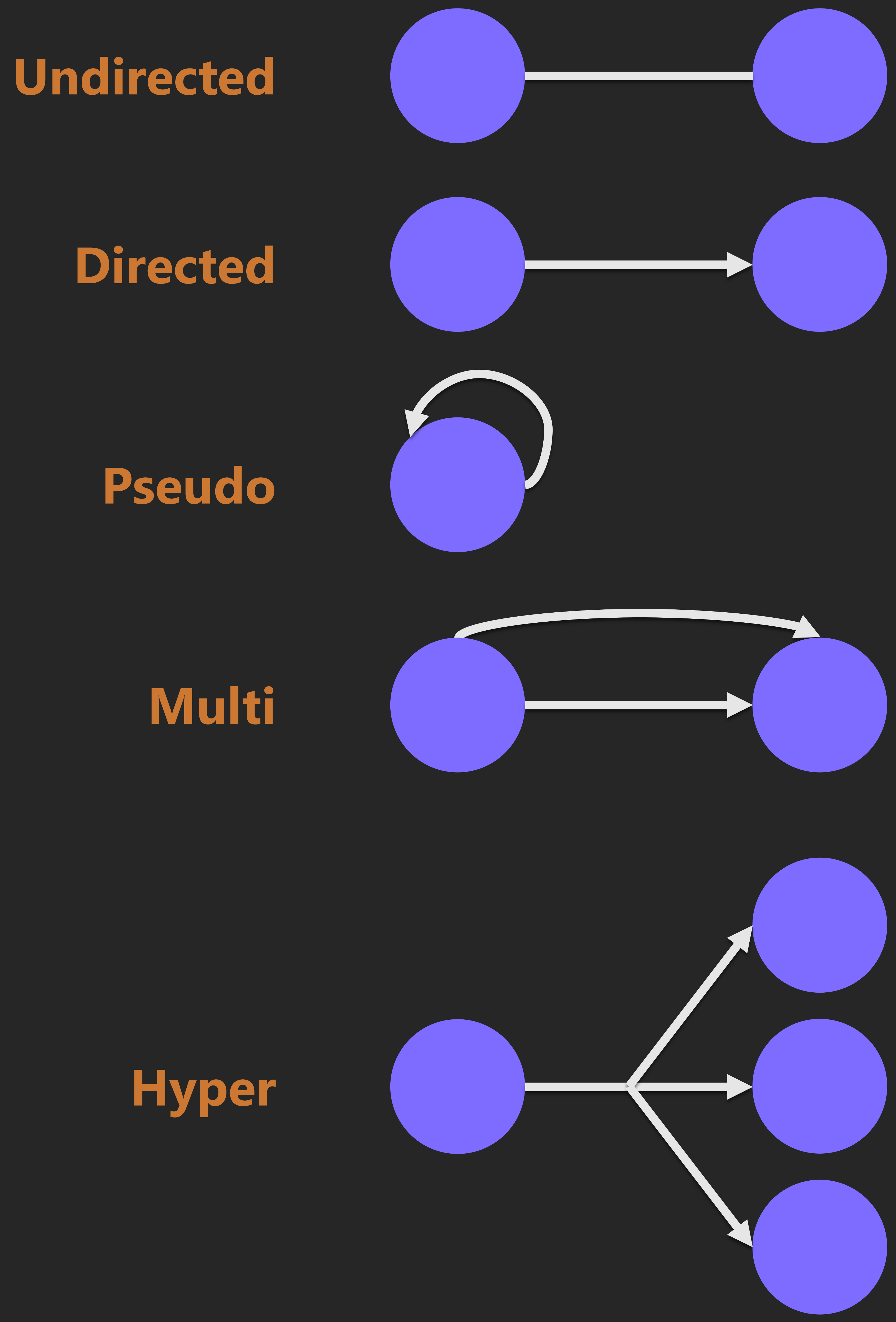
What is a graph?

What is a graph?

An abstract representation of a set of objects where some pairs are connected by links.

● Vertex (Node, Point, Instance)

→ Edge (Link, Line, Arc, Relation)



Undirected

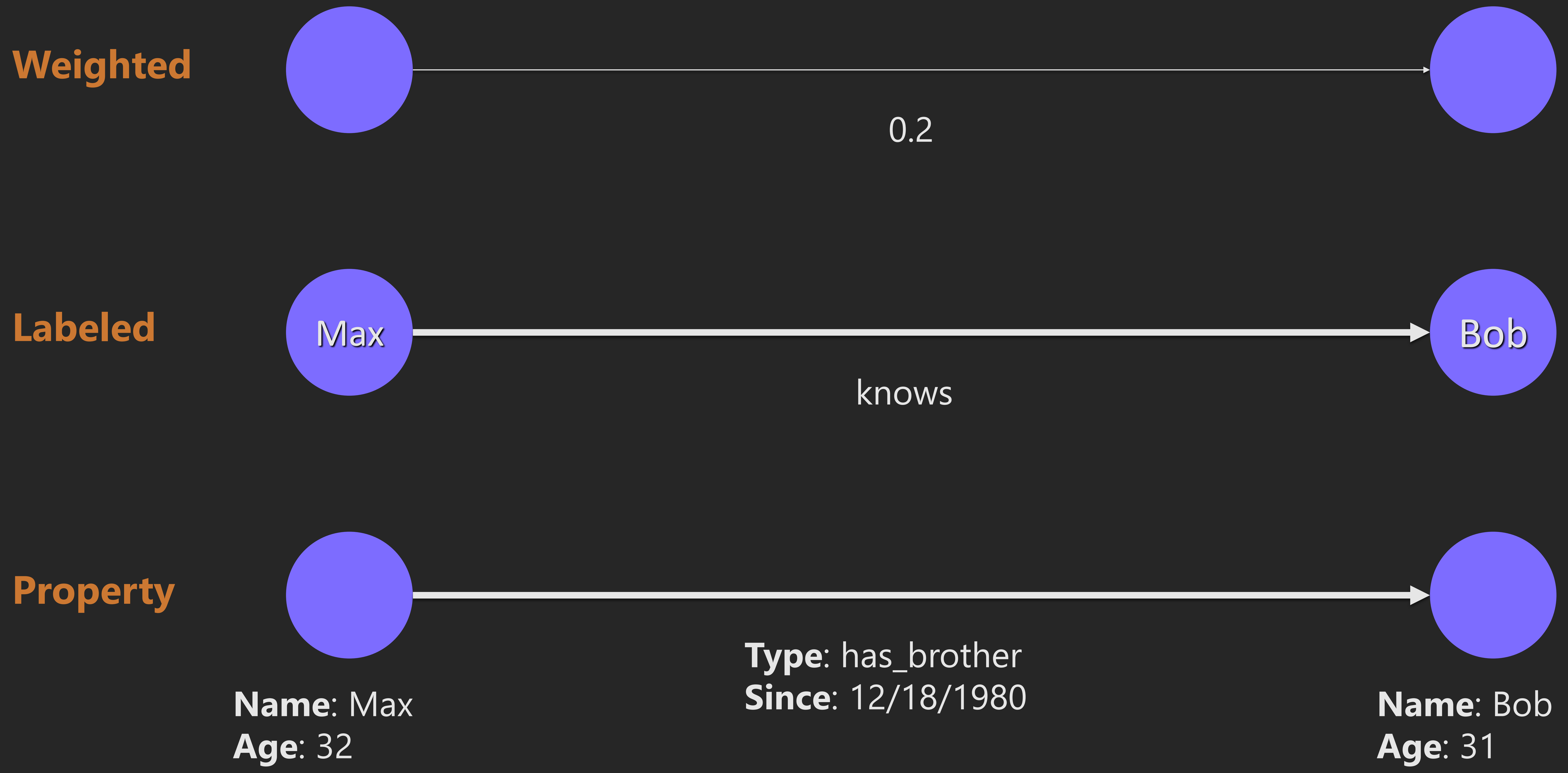
Directed

Pseudo

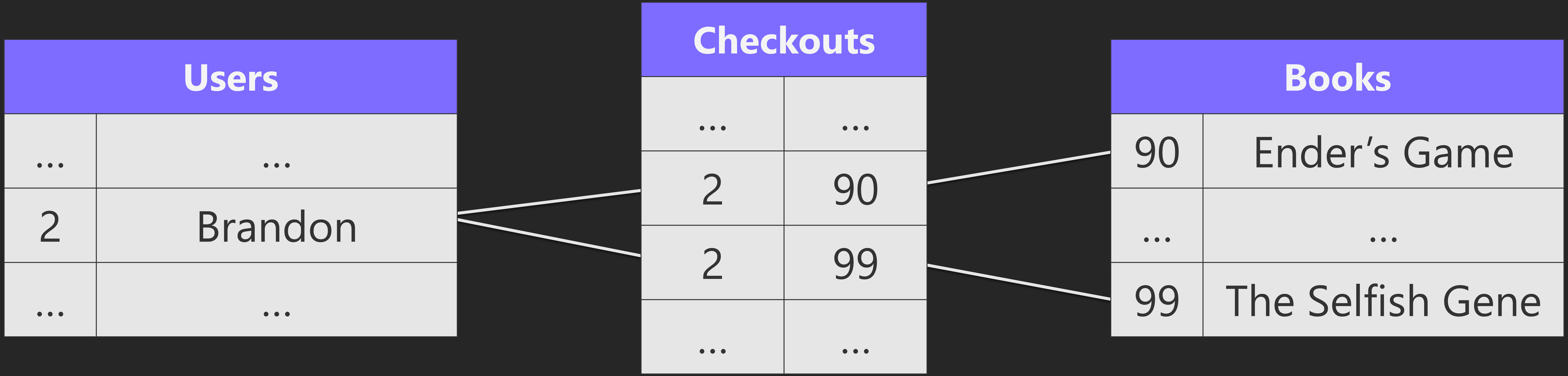
Multi

Hyper

What is a graph?



What is a graph?

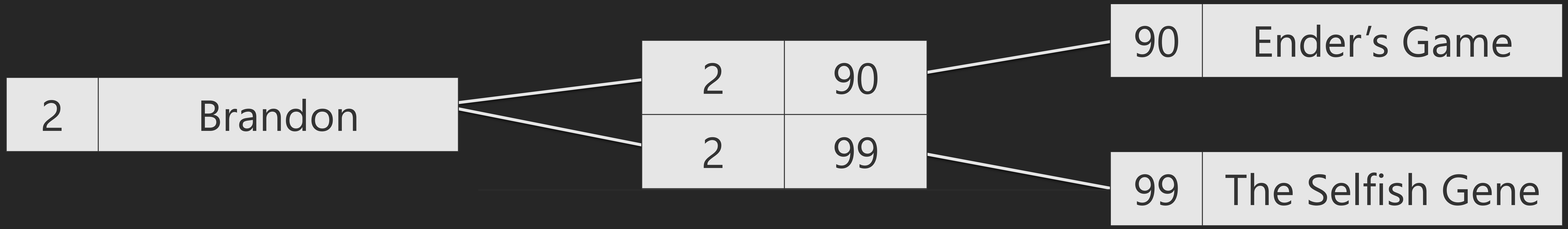


Users	
…	…
2	Brandon
…	…

Checkouts	
…	…
2	90
2	99
…	…

Books	
90	Ender’s Game
…	…
99	The Selfish Gene

What is a graph?

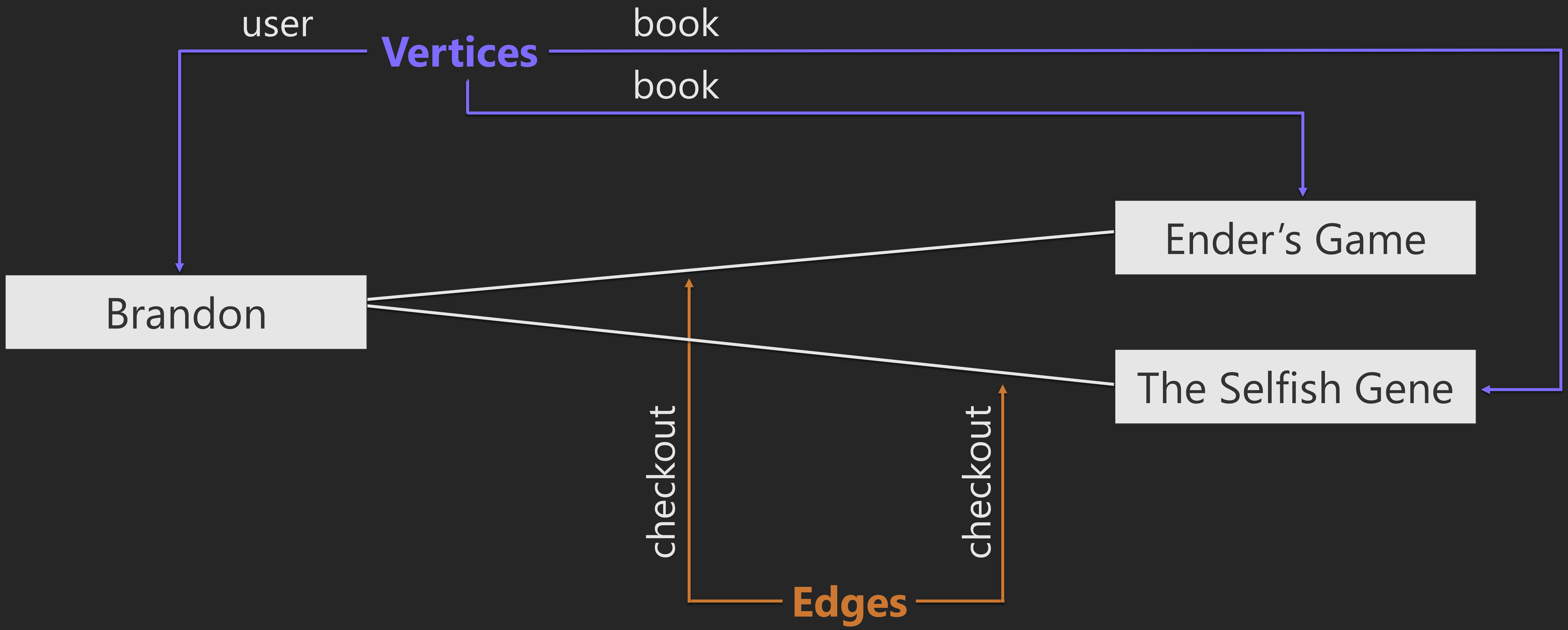


2	Brandon

2	90
2	99

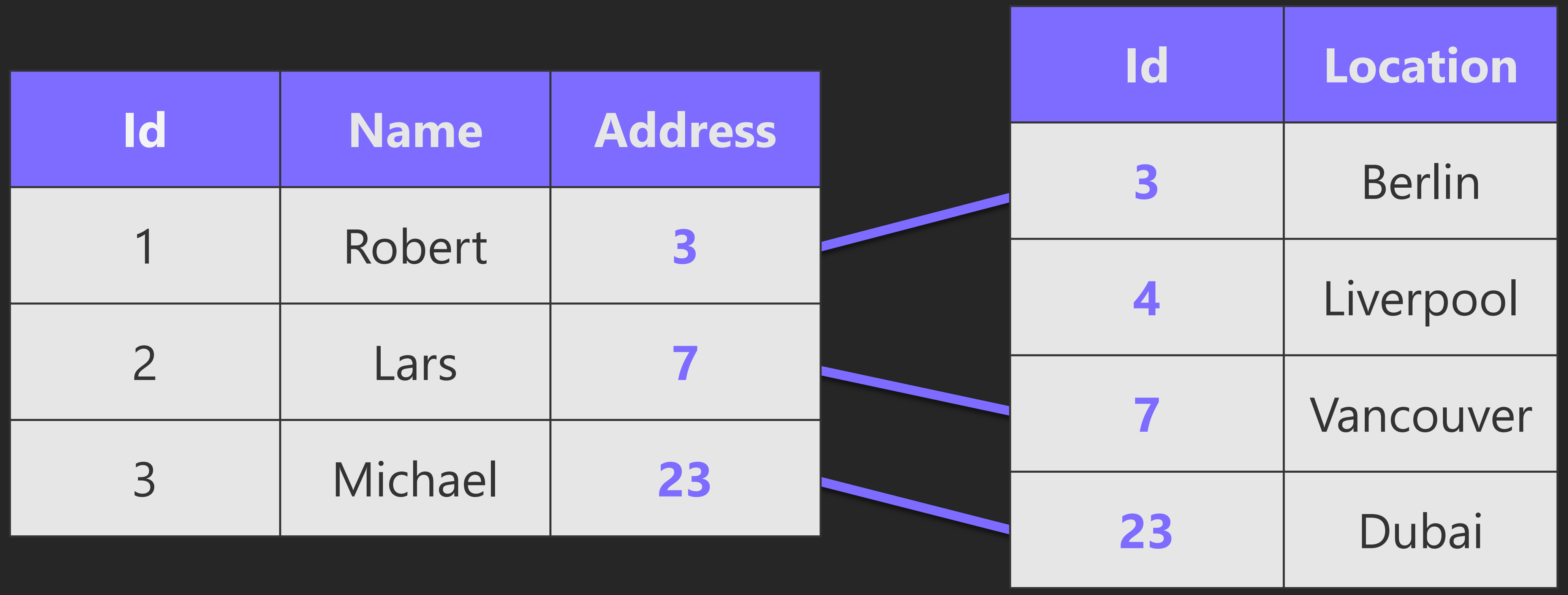
90	Ender’s Game
99	The Selfish Gene

What is a graph?



What is a graph?

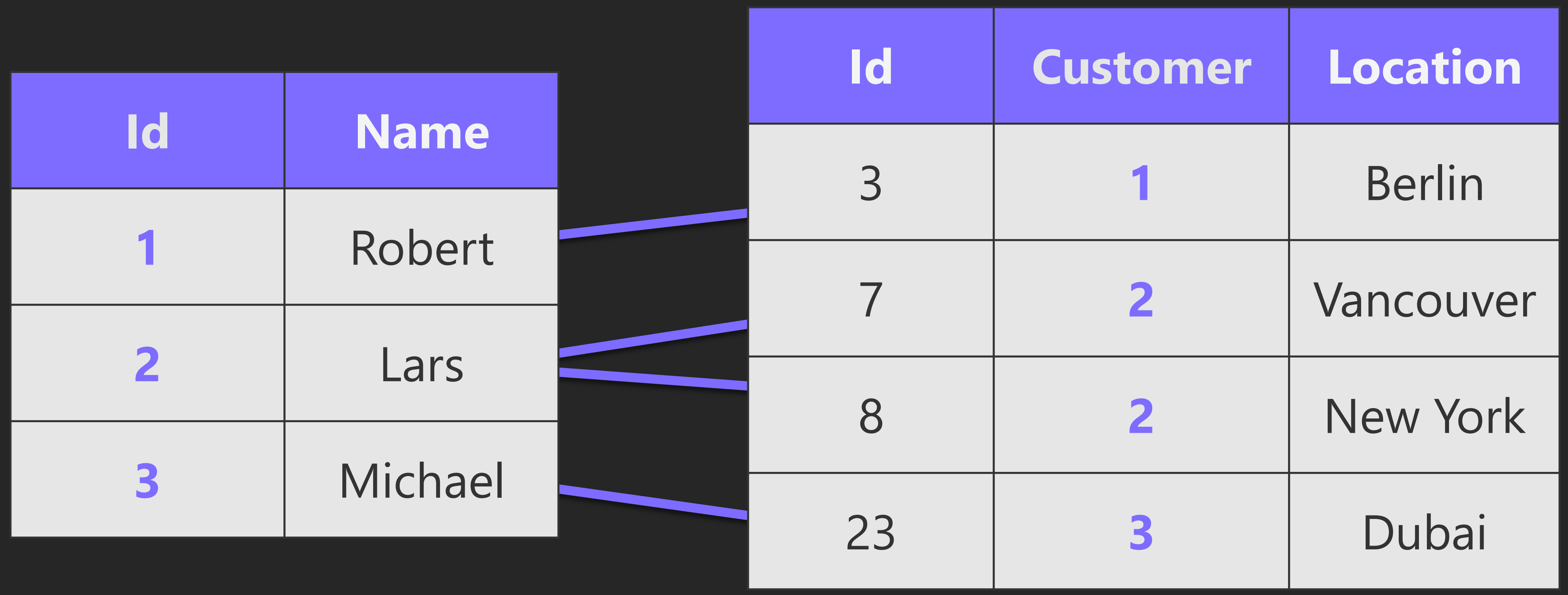
1:1 Relationship



Id	Name	Address
1	Robert	3
2	Lars	7
3	Michael	23

Id	Location
3	Berlin
4	Liverpool
7	Vancouver
23	Dubai

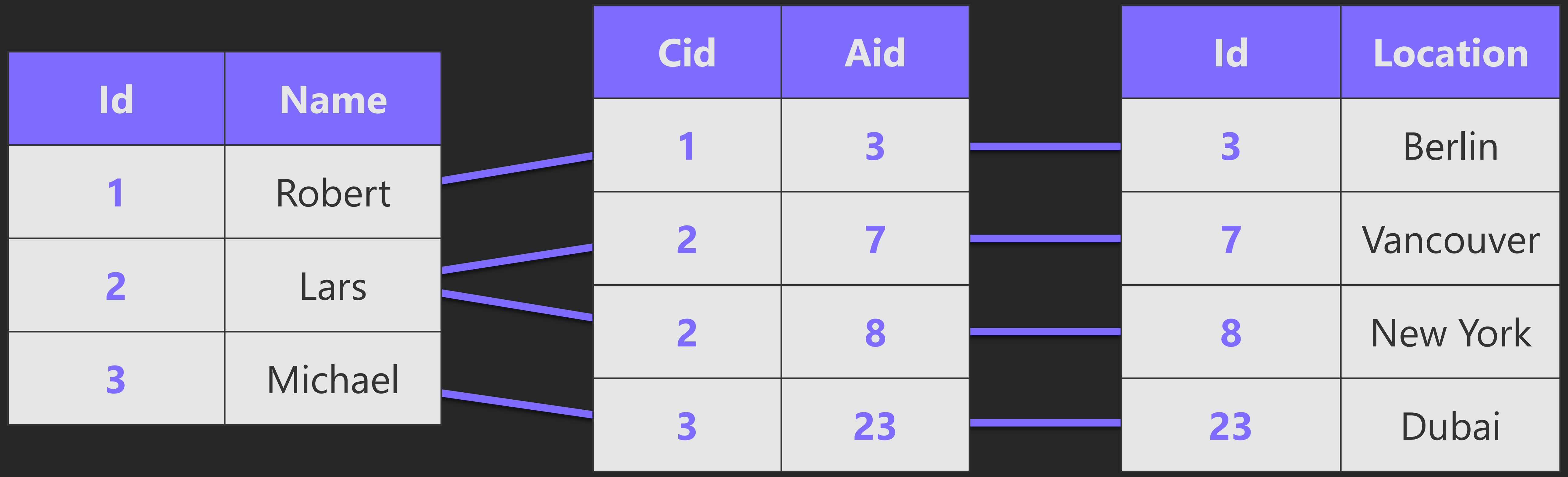
1:n Relationship



Id	Name
1	Robert
2	Lars
3	Michael

Id	Customer	Location
3	1	Berlin
7	2	Vancouver
8	2	New York
23	3	Dubai

m:n Relationship



Id	Name
1	Robert
2	Lars
3	Michael

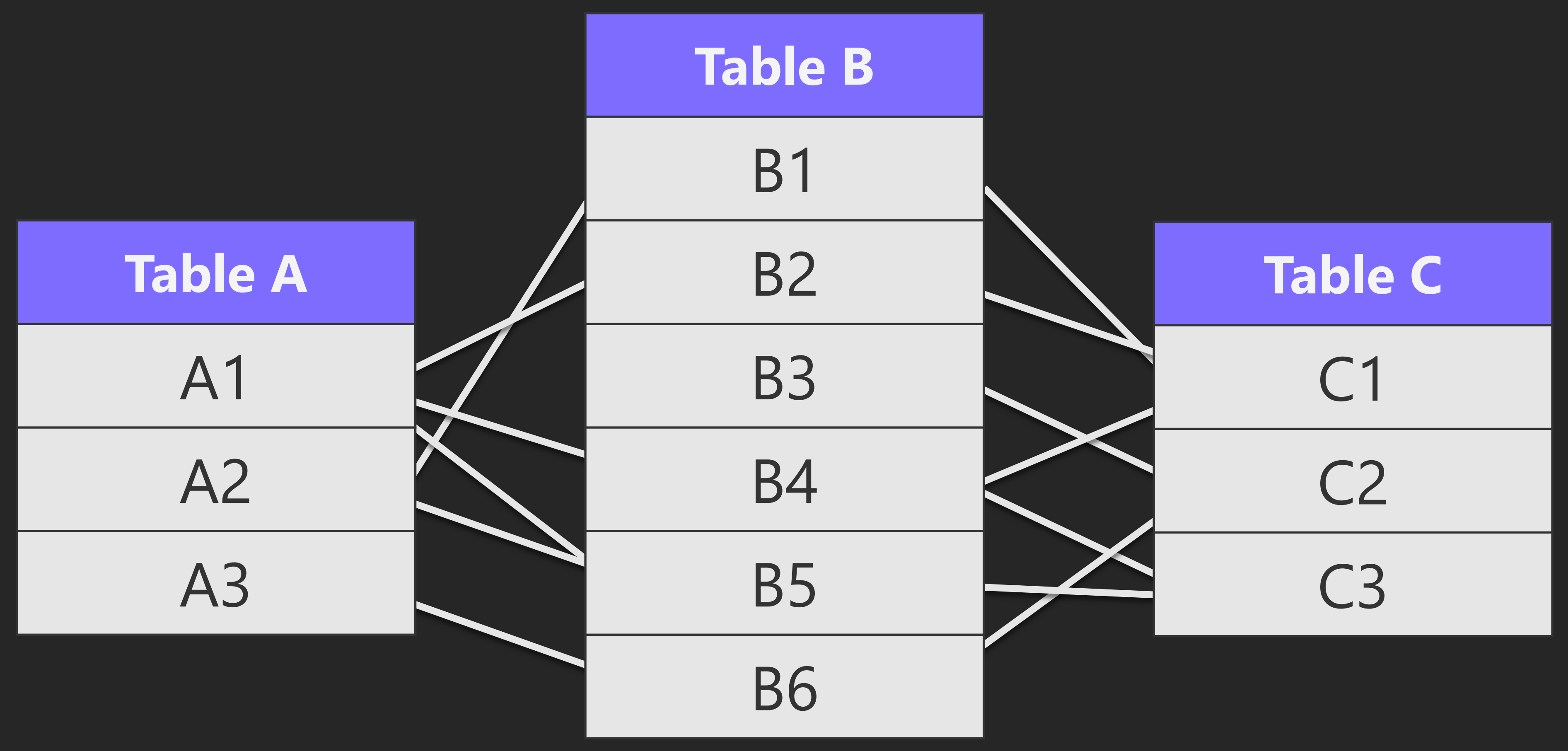
Cid	Aid
1	3
2	7
2	8
3	23

Id	Location
3	Berlin
7	Vancouver
8	New York
23	Dubai

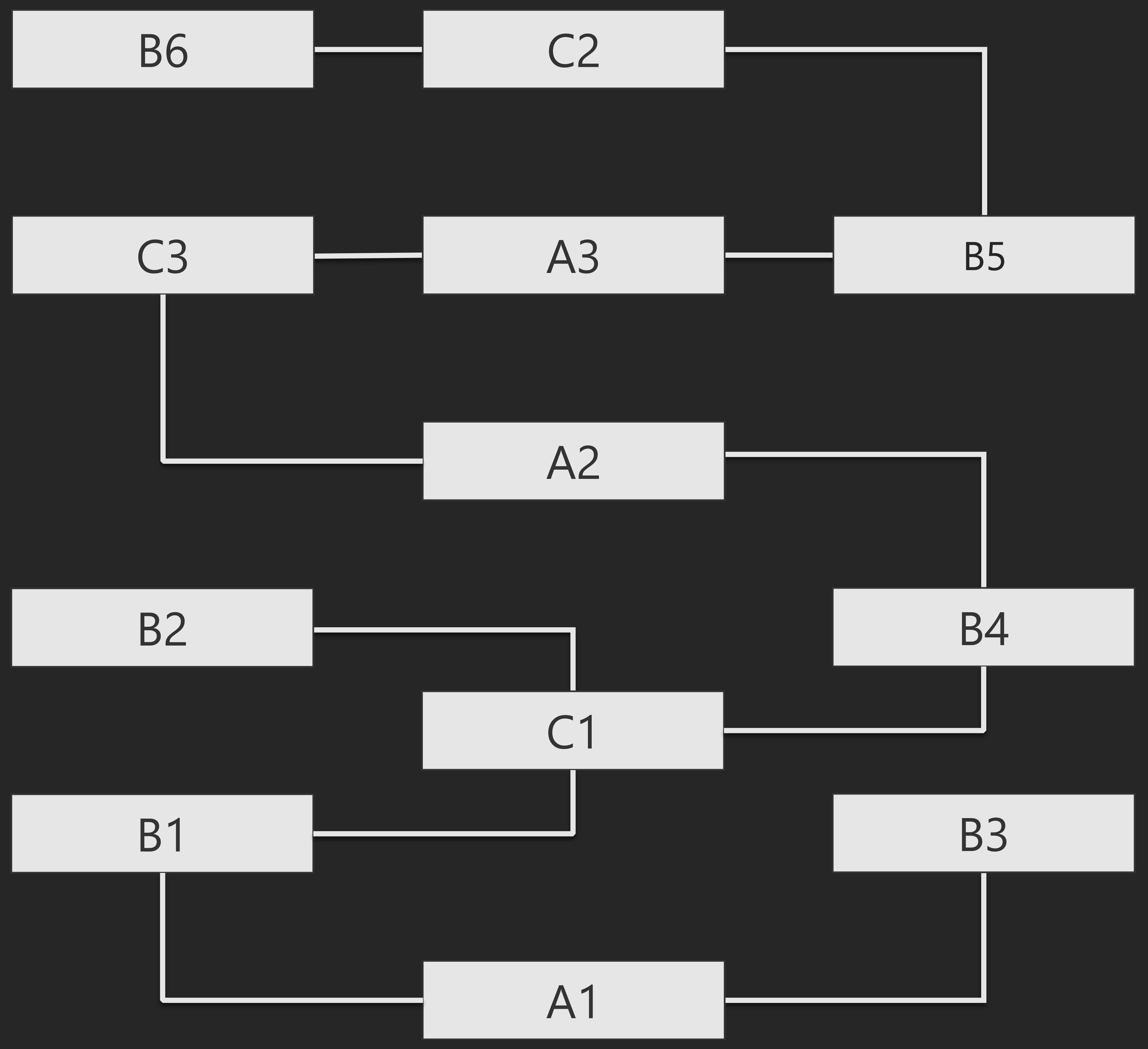
Why use graphs?

Why use graphs?

Optimized for aggregate data



Optimized for connected data



Why use graphs?

Use graphs when:

- Problems with join performance
- Joining more than 7 tables together
- The majority of your tables are junction tables
- Written stored procedures with multiple recursive self and inner joins
- Continuously evolving data set (often involves wide and sparse tables)
- The shape of the domain is naturally a graph
- Constantly changing schema

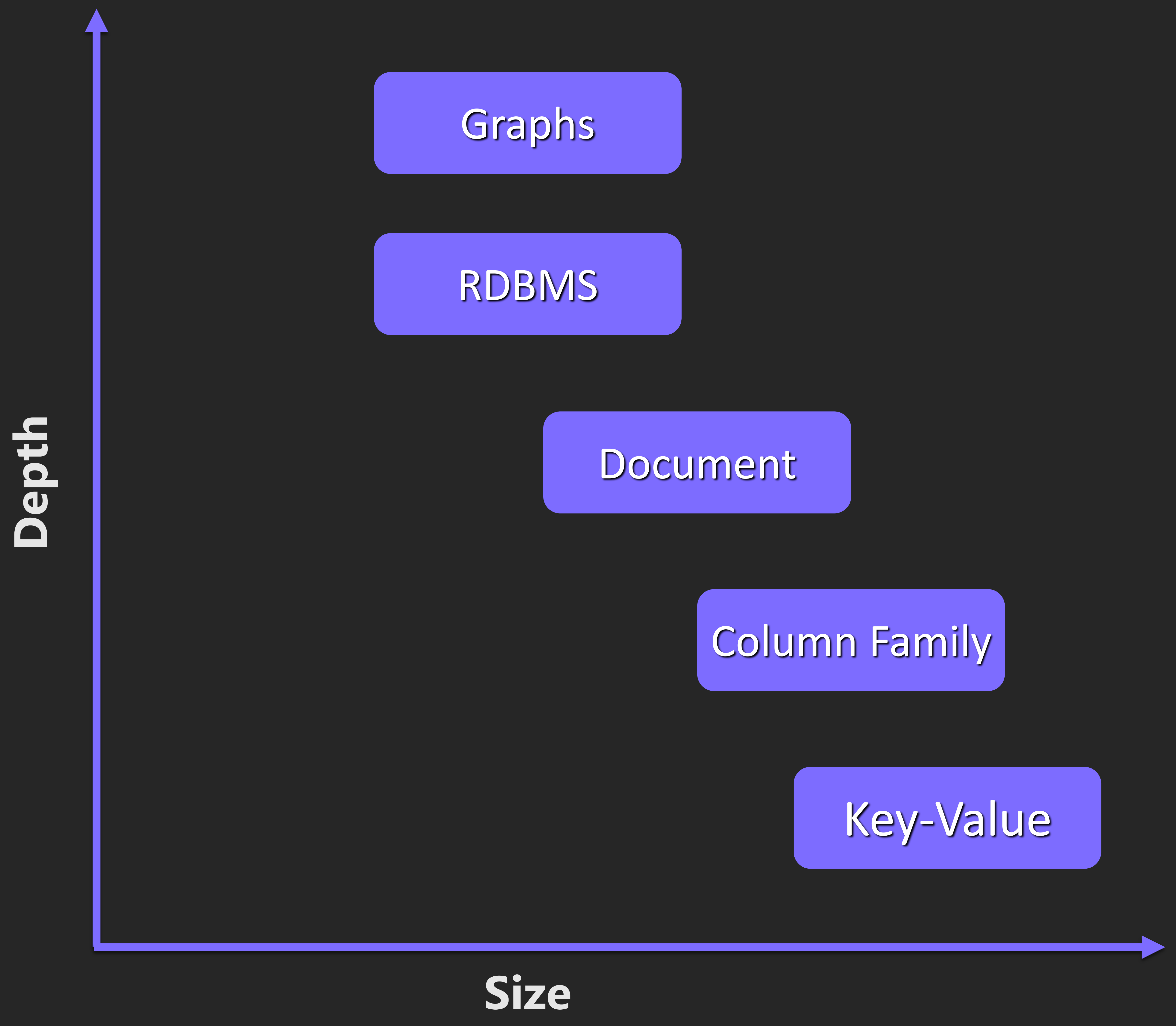
Graphs are good at:

- Path finding (how do people know each other)
- Highly connected data (social networks)
- Recommendations (e-commerce)
- A* (least-cost path analysis)

Graphs are designed to:

- Store interconnected data
- Make it easy to make sense of data connections
- Make it easy to evolve the database
- Enable high-performance on operations for:
  - Discovery of connected data patterns
  - Relatedness queries > depth 1

Why use graphs?



Graphs

RDBMS

Document

Column Family

Key-Value

Depth

Size

Friends-of-friends
1,000,000 people
~50 friends each

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110k
4	1543.505	1.359	~600k
5	Unfinished	2.132	~800k

Friends-of-friends
~50 friends each
Depths of 4

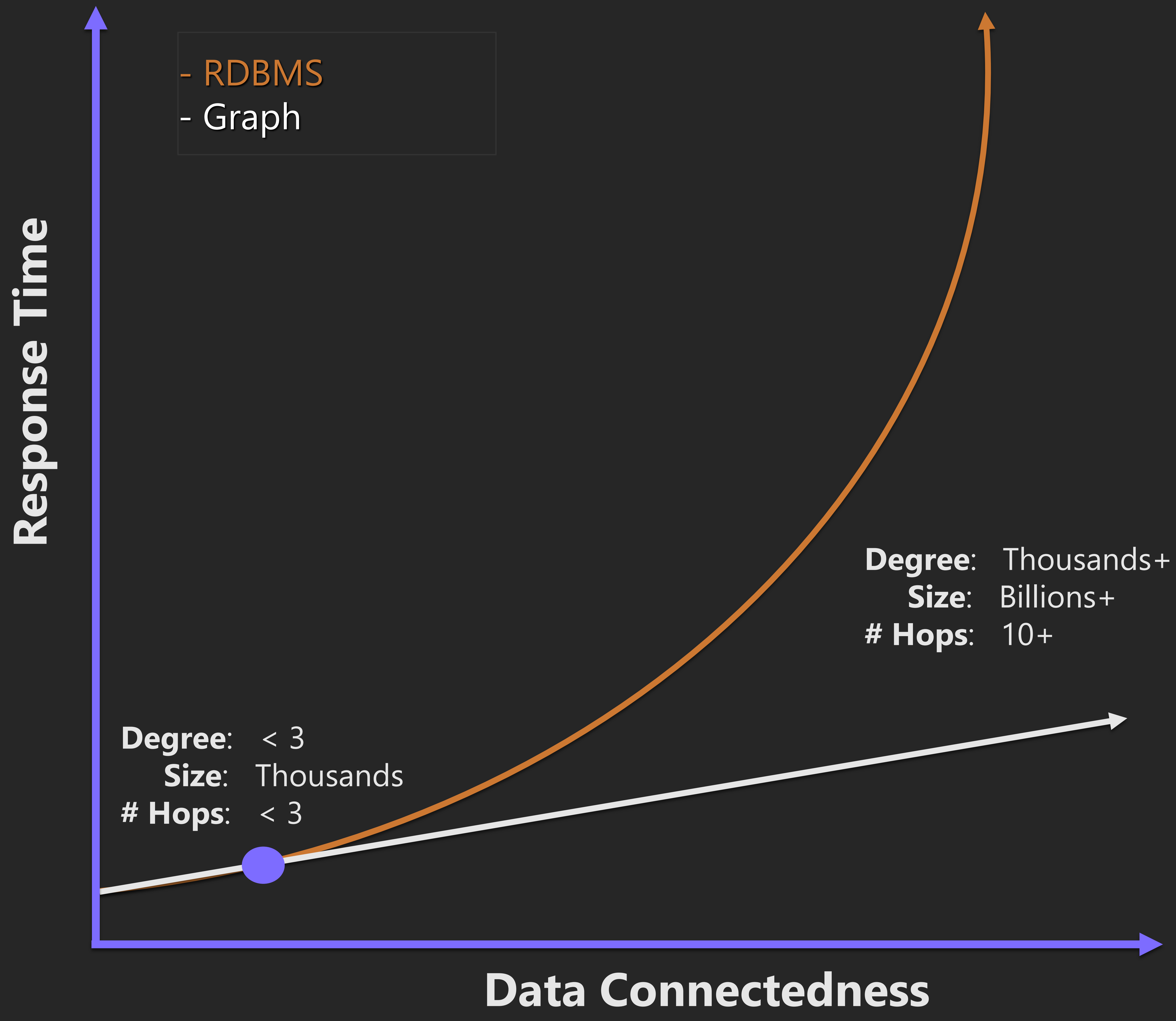
Database	# persons	Query time
MySQL	1,000	2 sec
Neo4j	1,000	2 ms
Neo4j	1,000,000	2 ms

Why use graphs?

Query Response Time

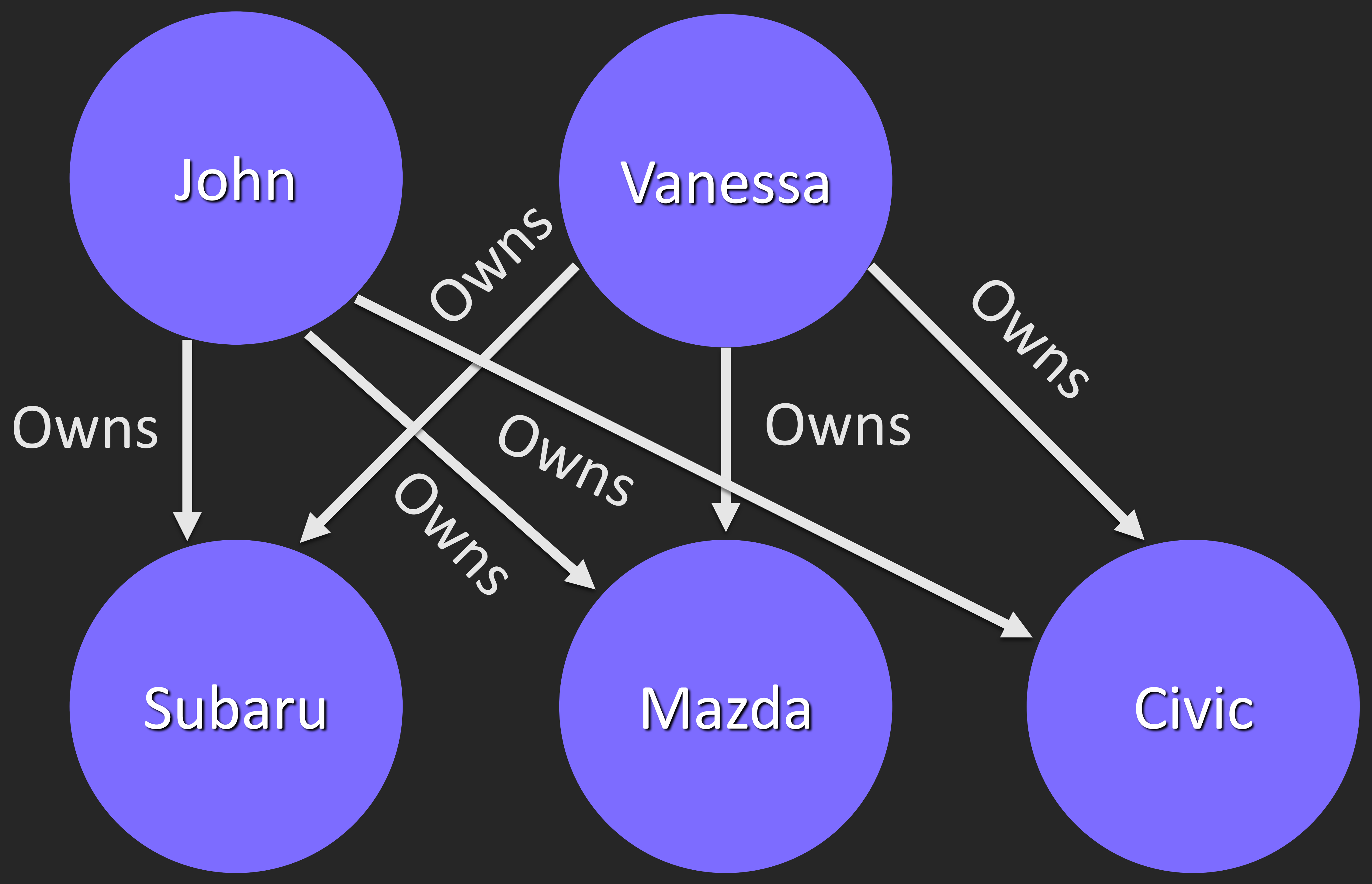
= **f**(graph density, graph size, query degree)

- Graph density (avg # rel’s / node)
- Graph size (total # of nodes in the graph)
- Query degree (# of hops in one’s query)

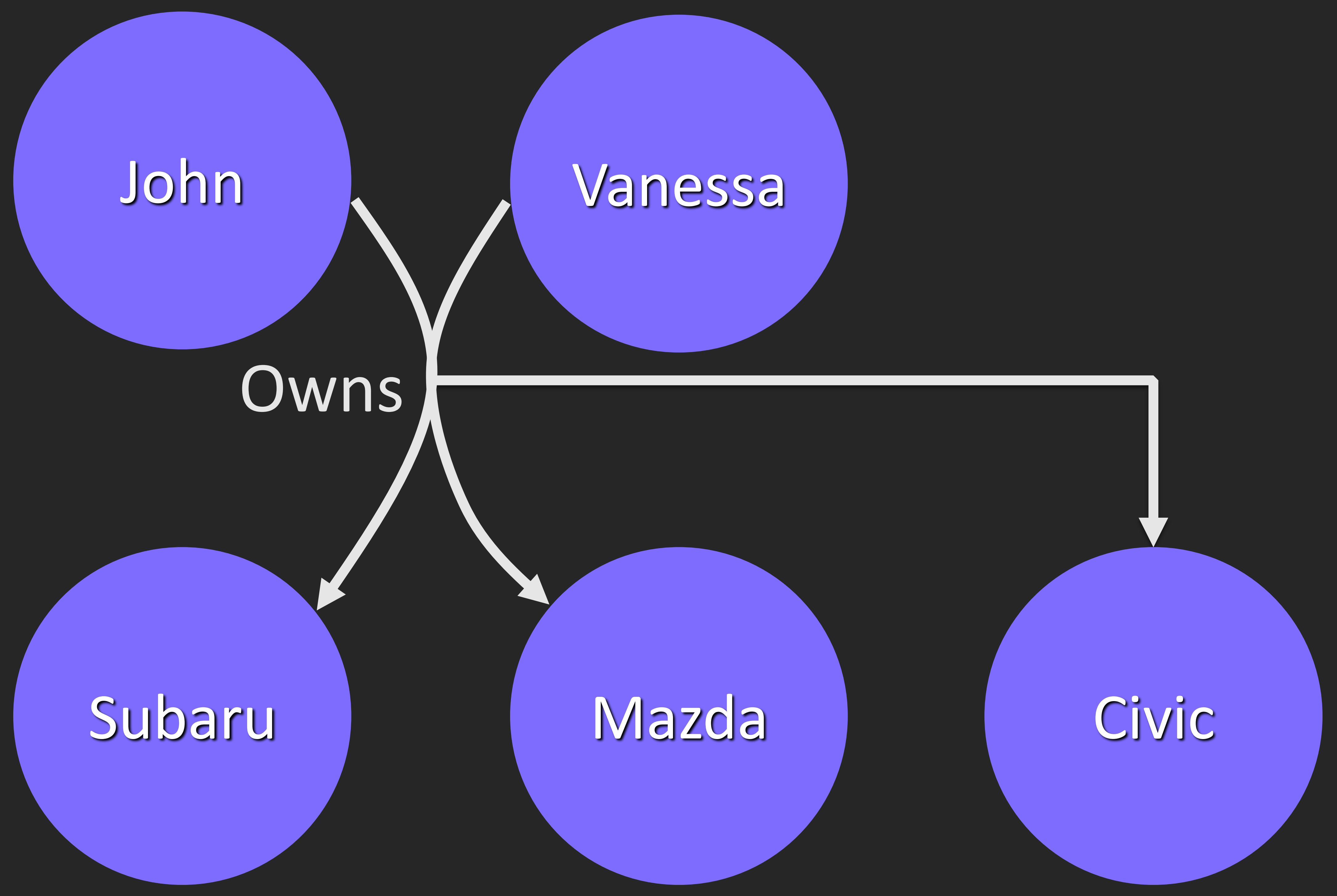


Are hypergraphs useful?

Are hypergraphs useful?



Directed Graph

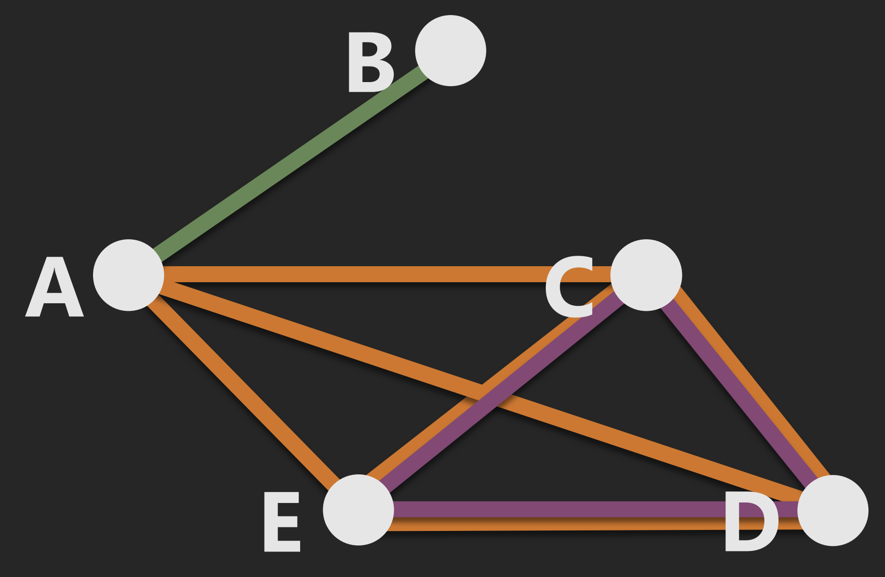


Hypergraph

Are hypergraphs useful?

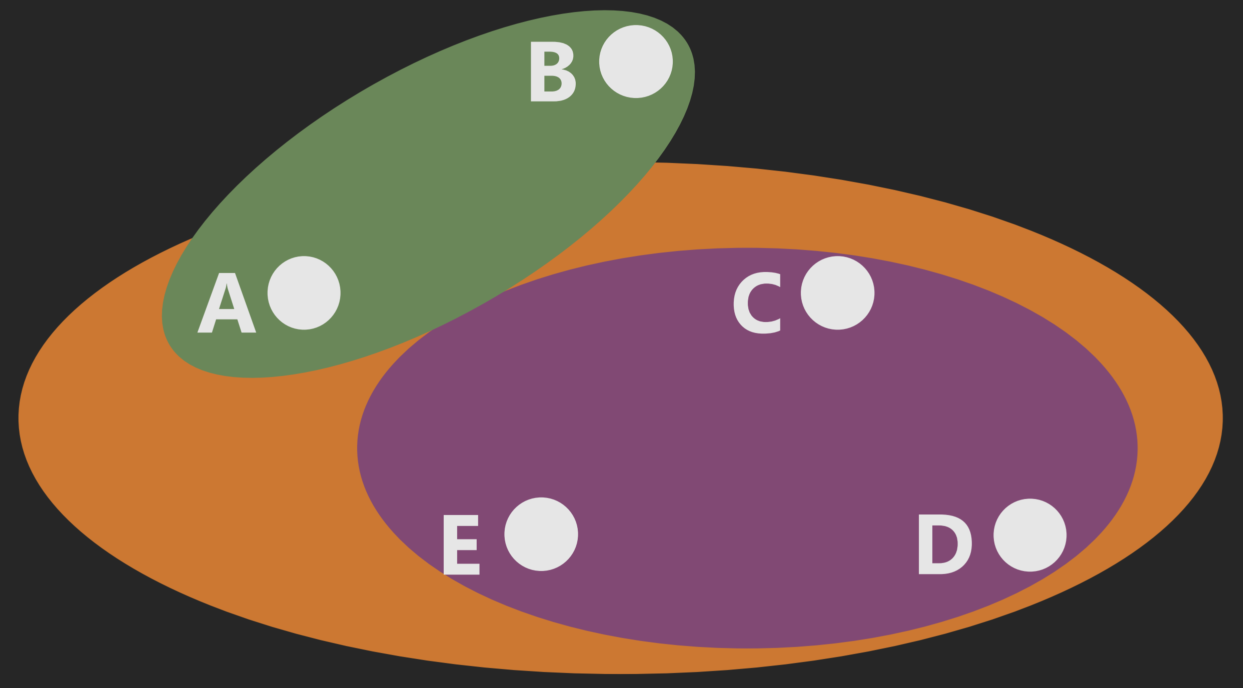
Graph

	1	2	3	4	5	6	7	8	9	10
A	X				X	X	X			
B	X									
C		X	X		X			X	X	
D		X		X		X		X		X
E			X	X			X		X	X



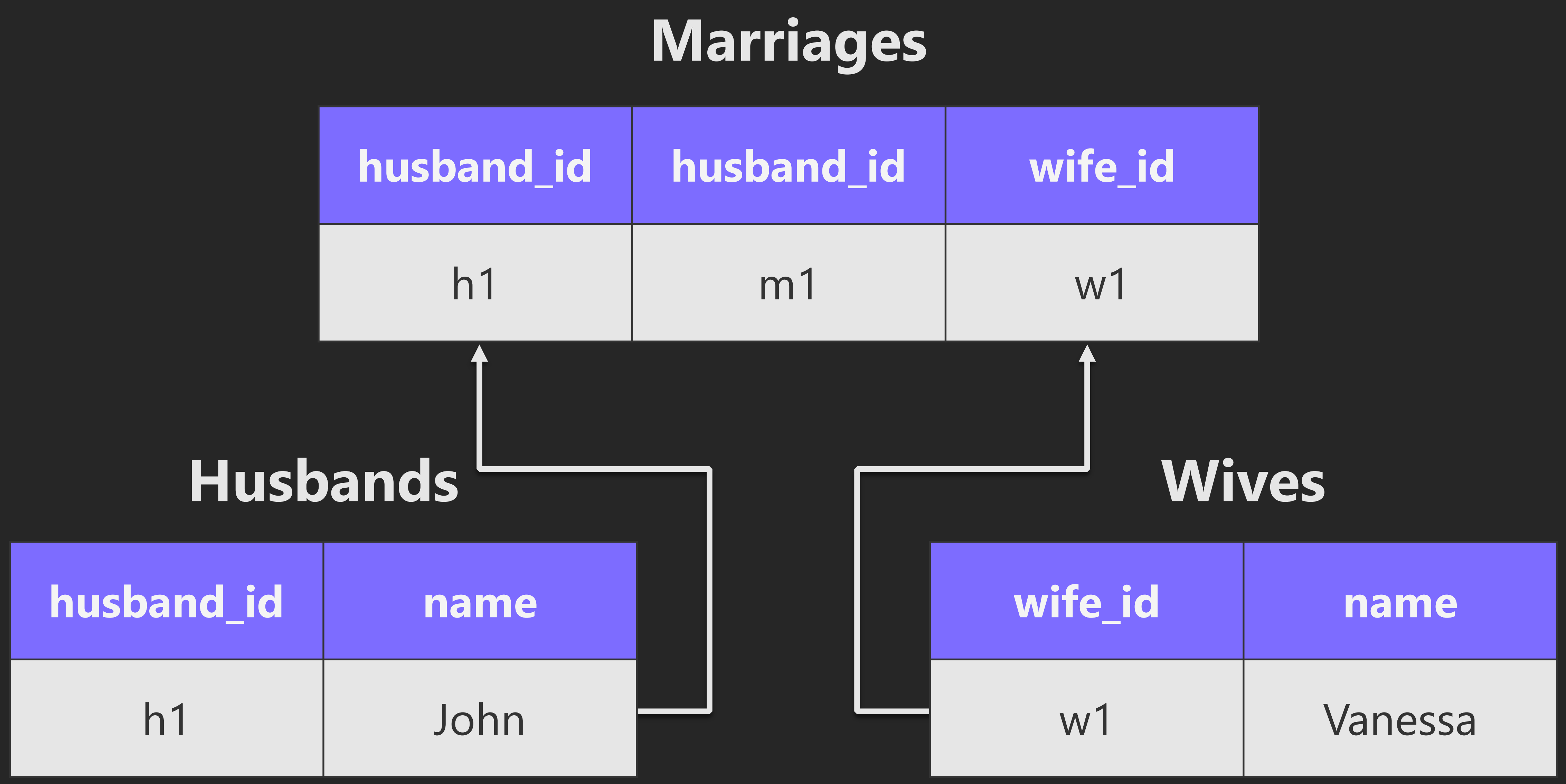
Hypergraph

	1	2	3
A	X		X
B	X		
C		X	X
D		X	X
E		X	X



Hypergraphs generalize the common notion of graphs by relaxing the definition of edges
Graph: Edge = pair of vertices
Hypergraph: Hyperedge = set of vertices

Scenario – Traditional Marriage



Marriages

husband_id	husband_id	wife_id
h1	m1	w1

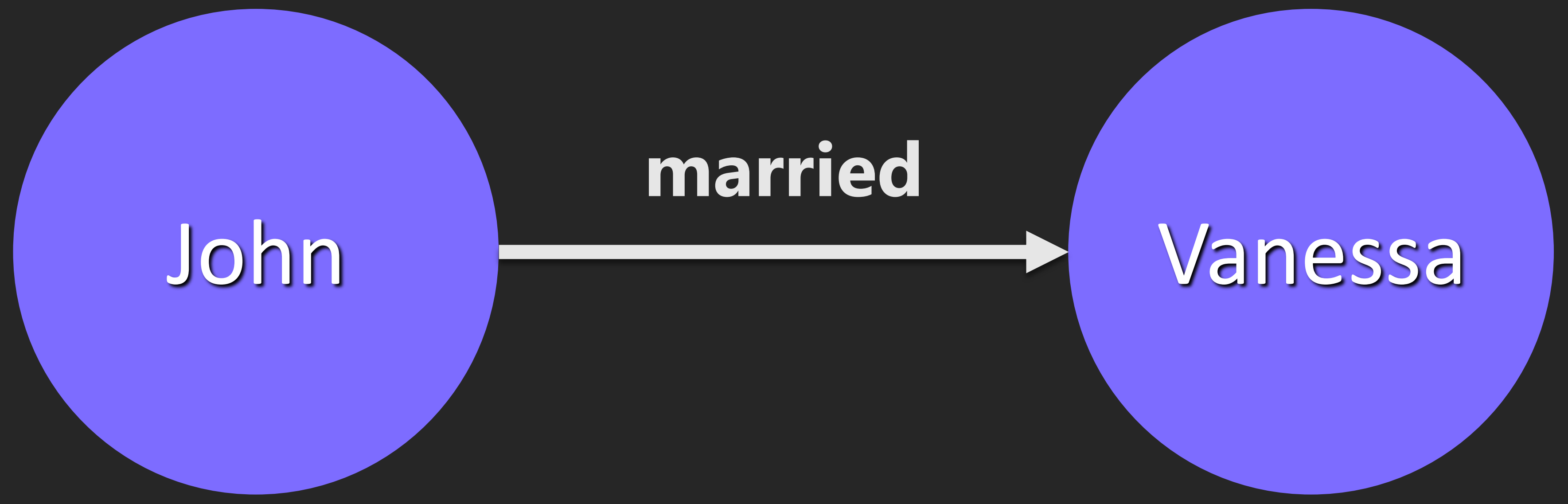
Husbands

husband_id	name
h1	John

Wives

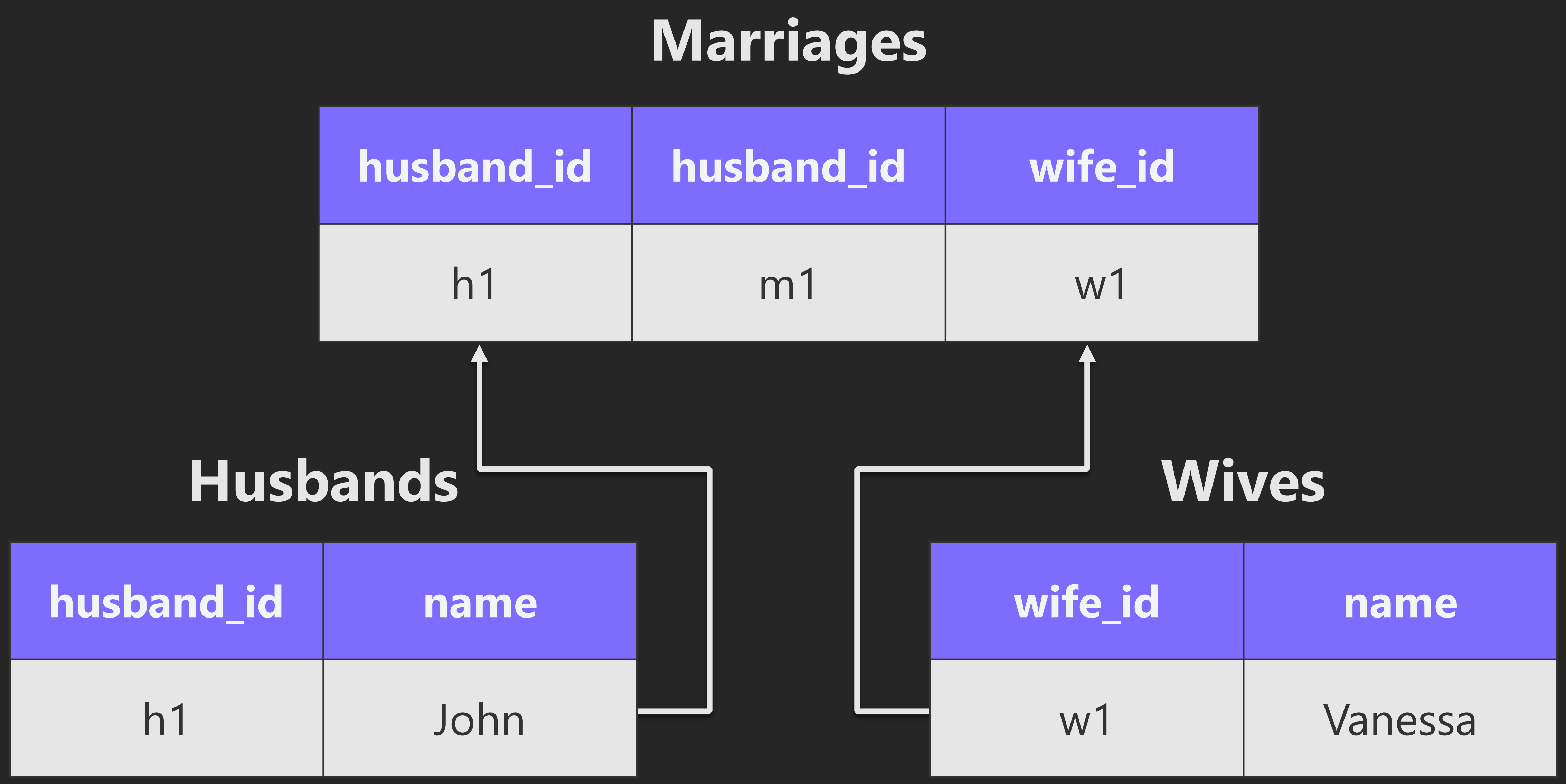
wife_id	name
w1	Vanessa

Relational



Directed

Scenario – Polygamous Marriage



Marriages

husband_id	husband_id	wife_id
h1	m1	w1

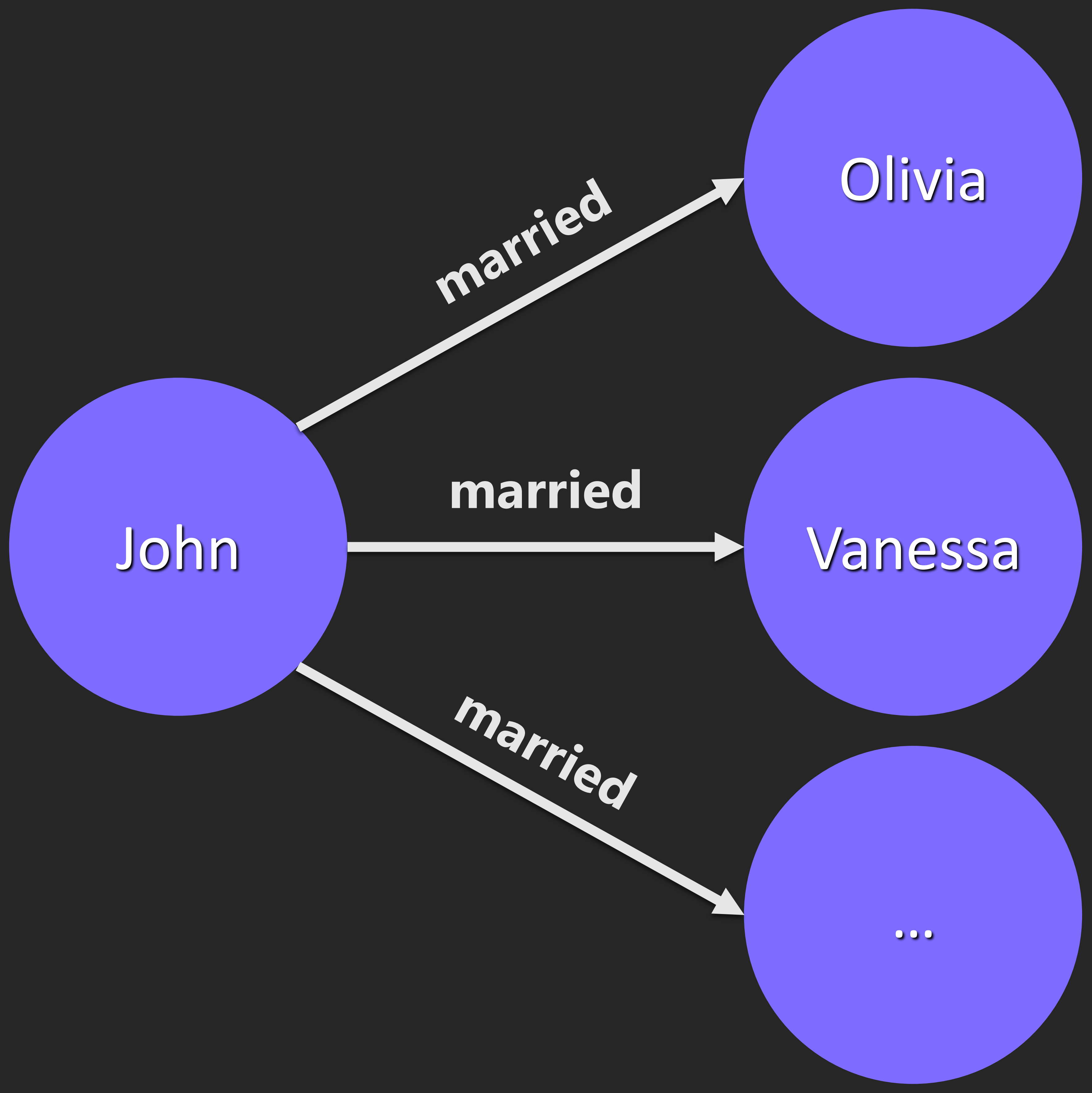
Husbands

husband_id	name
h1	John

Wives

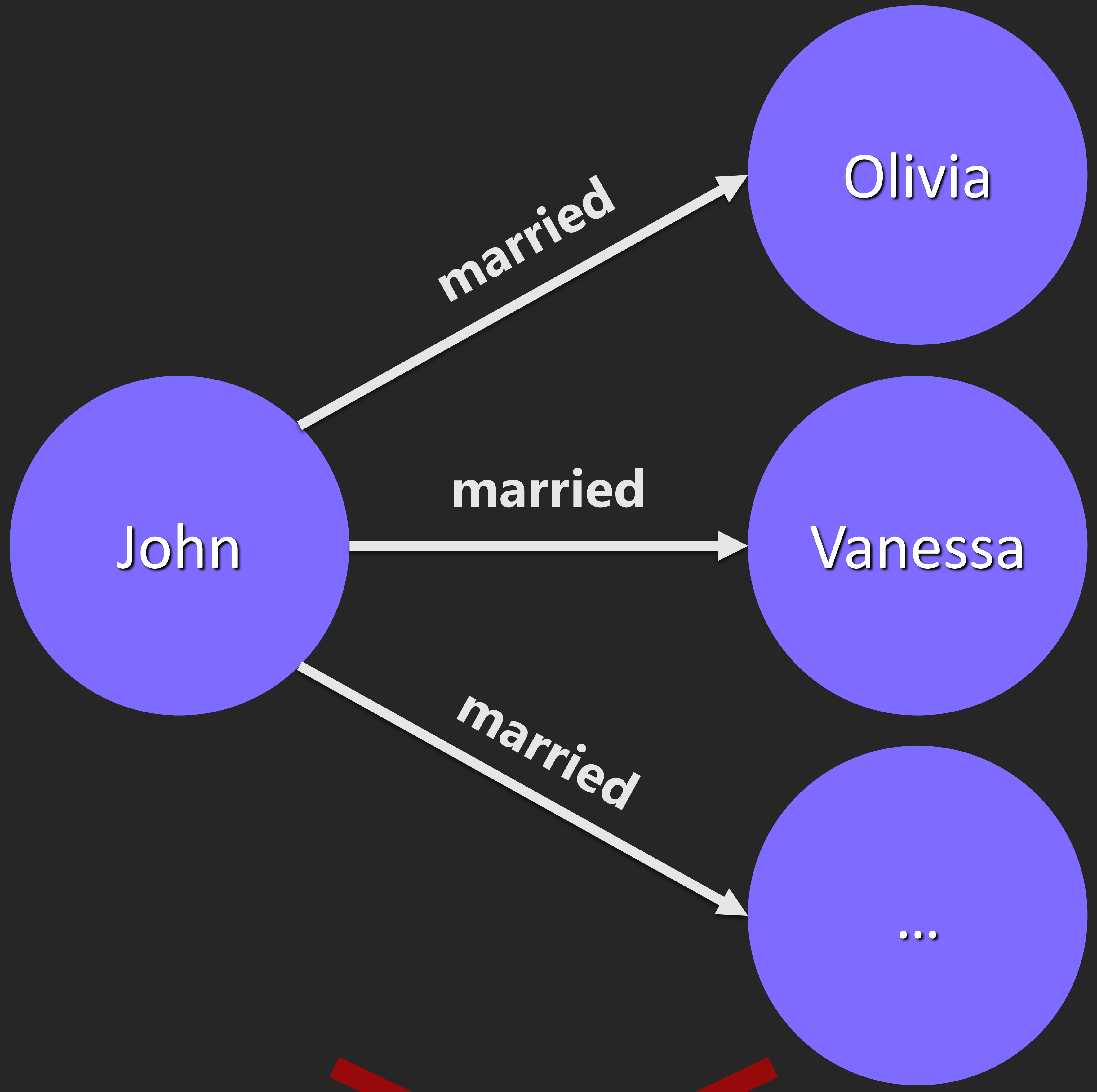
wife_id	name
w1	Vanessa

~~Relational~~

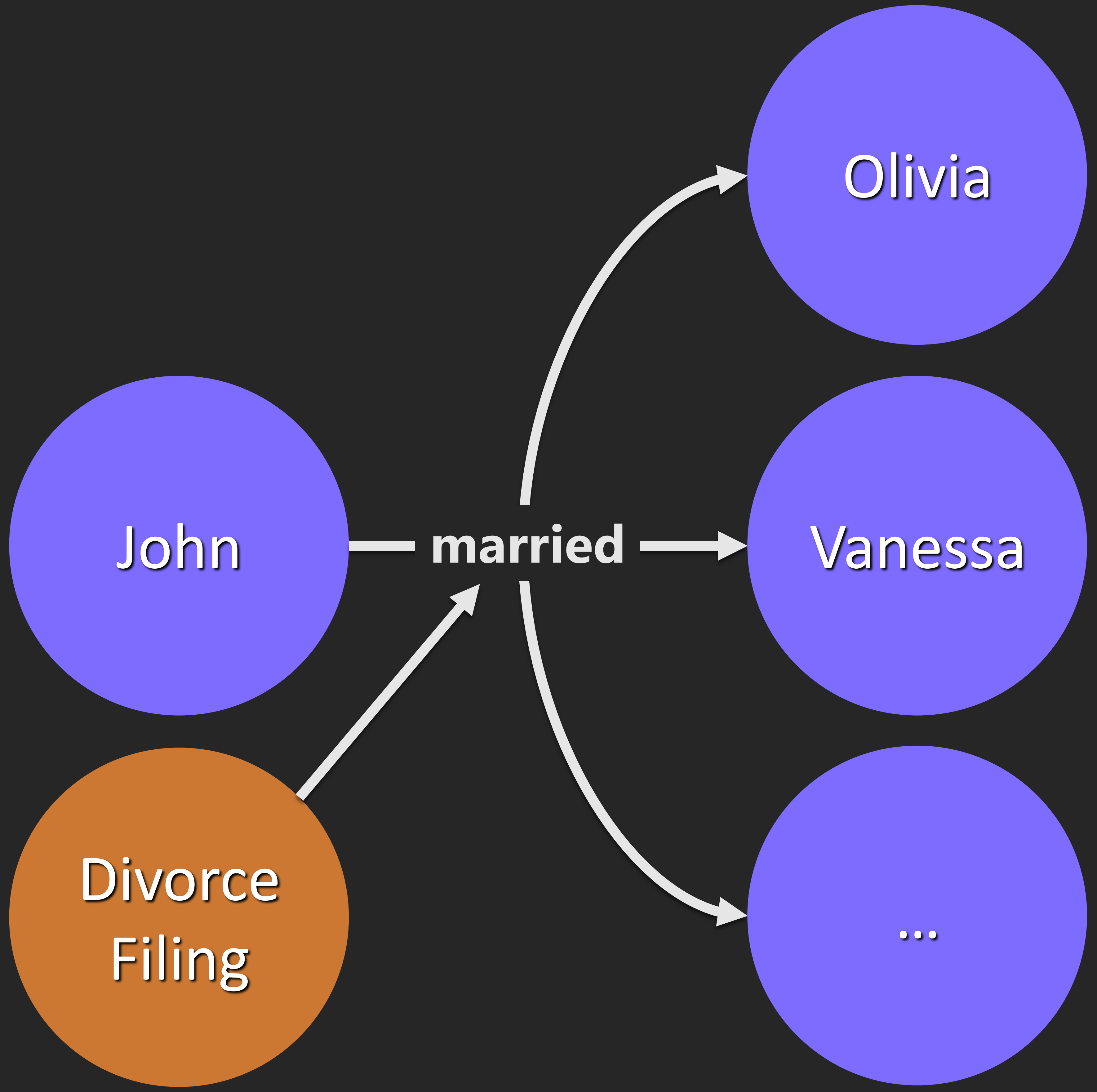


Directed

Scenario – Divorce Filing



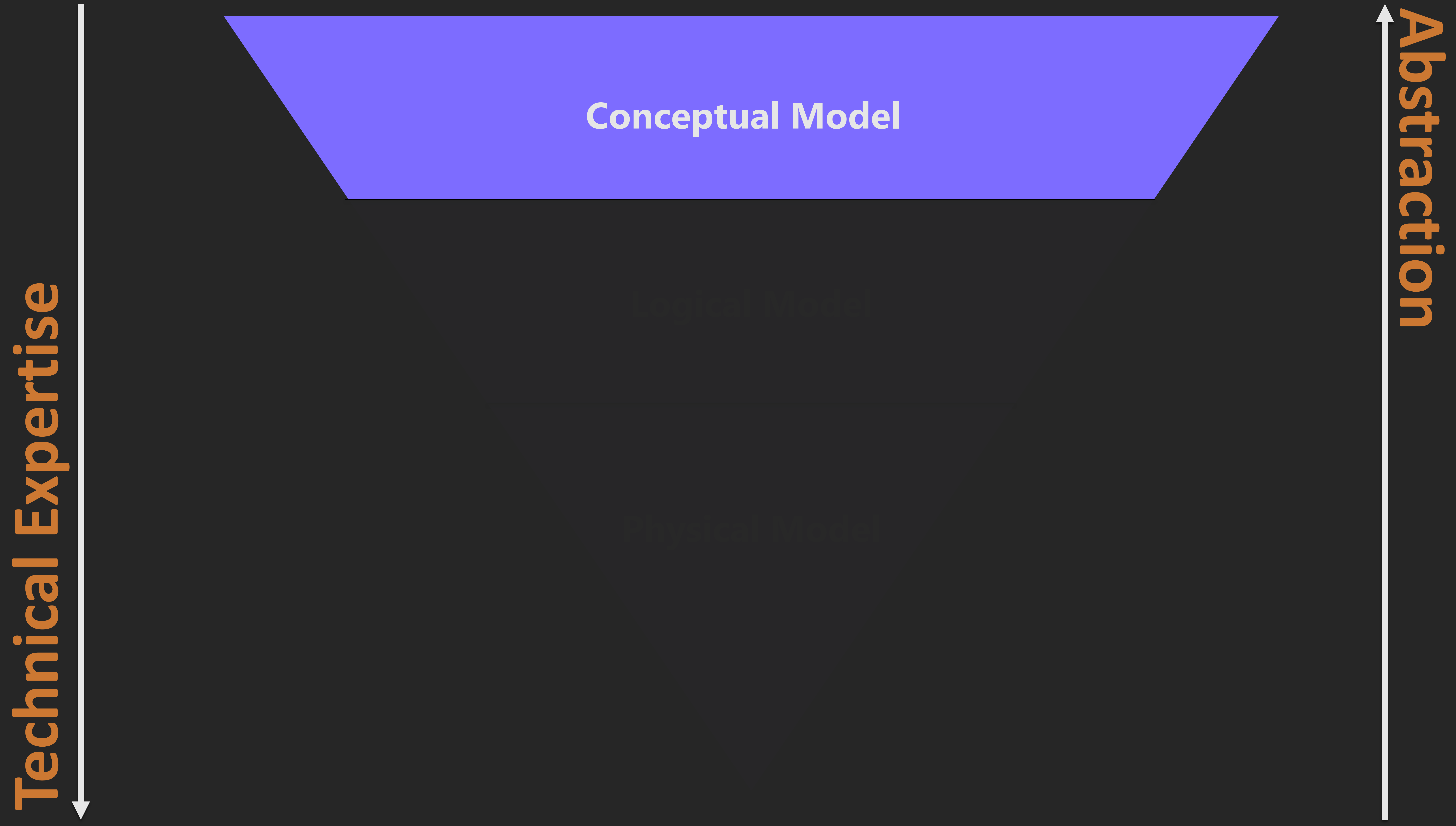
~~Directed~~



Hyper

How do I model a graph?

Data Modeling



Conceptual Model

Entities

- Attribute value comprises a **complex value** type (e.g. address)
- Values with conceptual identities
- Value requires qualification via relation

- Example:
  - Find all recent orders delivered to the **same delivery address** (complex value type)

Relations

- Specify **weight**, **strength**, or some other **quality** about the relationship

- Example:
  - Find all my colleagues who are **level 2 or above** (relationship quality) in a **skill** (attribute value) we have in common

Attributes

- There's no need to qualify the relationship and consists of a **simple value** type (e.g. color)
- Have no conceptual identity (metadata)

- Example:
  - Find those projects written by contributors to my projects that use the same **language** (attribute value) as my projects

Determine Entities

Which library users have books currently lent which are over-due?

Determine Entities

Which library users have books currently lent which are over-due?

Determine Relations

Which library users have books currently lent which are over-due?

Determine Relations

Which library users have books currently lent which are over-due?

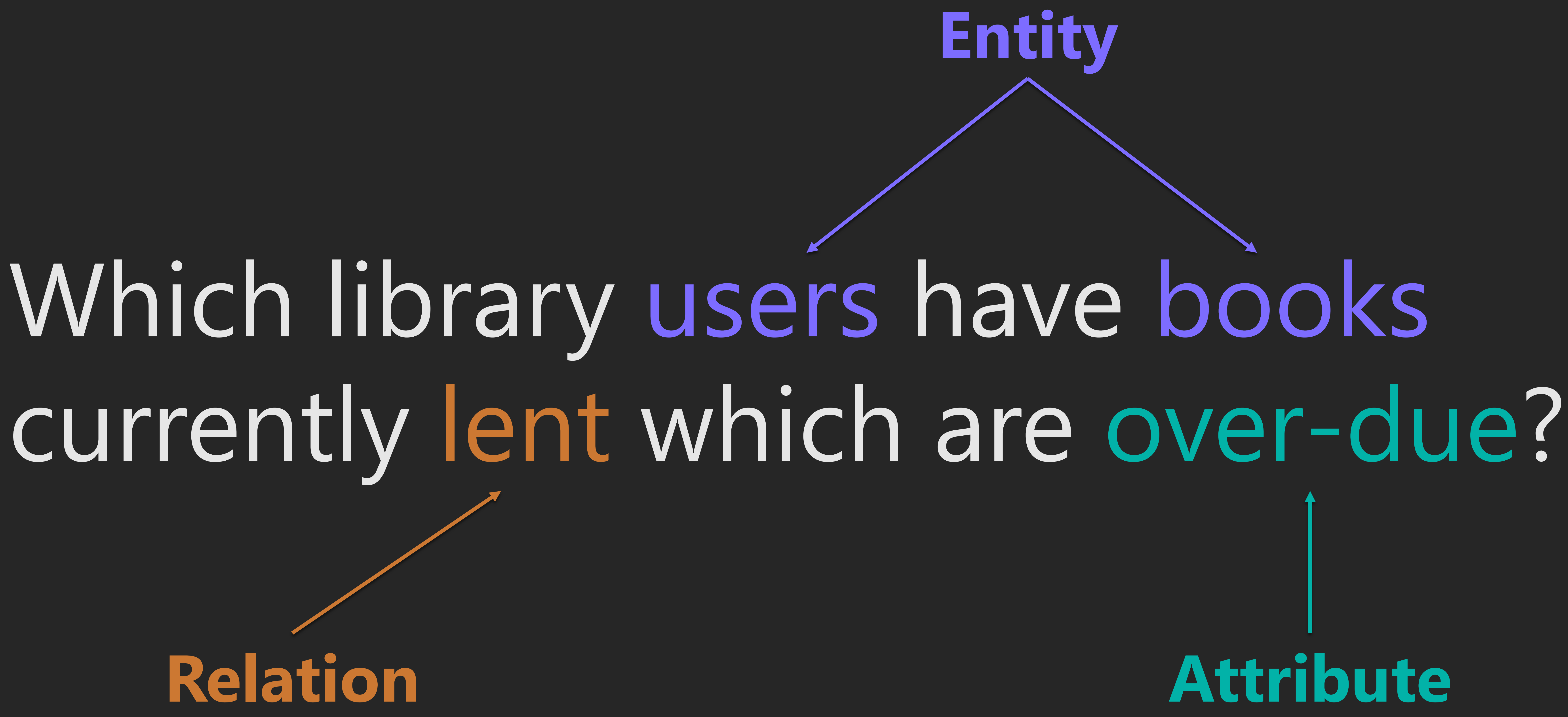
Determine Attributes

Which library users have books currently lent which are over-due?

Determine Attributes

Which library users have books currently lent which are over-due?

Conceptual Model



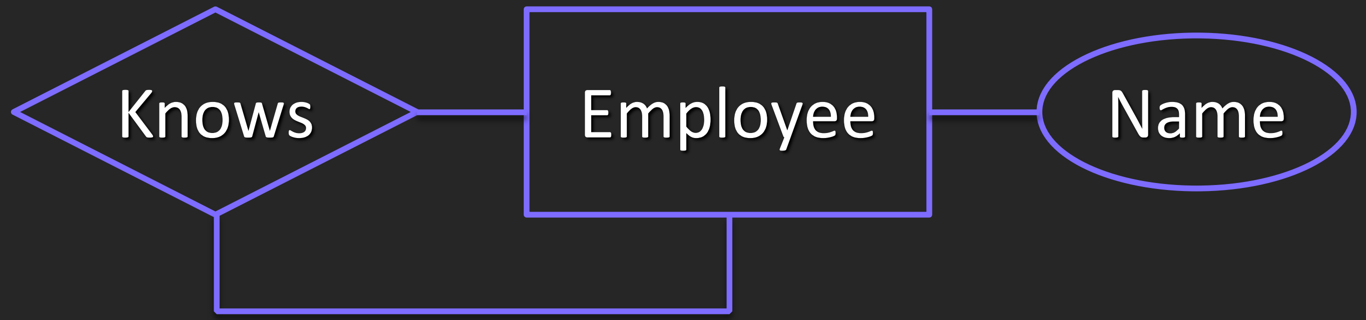
Entity

Which library users have books currently lent which are over-due?

Relation

Attribute

Entity–Relationship Model

Symbol	Meaning	Example
▭	Entity	Employee
⬭	Attribute	Name
◇	Relationship	Knows
―	Link	

Draw Conceptual Model

Objective

Which library users have books currently lent which are over-due?

Data Available

Book
- Author
- Title
- Publish date
- Lend date
- Due date

User
- First/middle/last name
- DOB
- Email
- Gender

Draw Conceptual Model

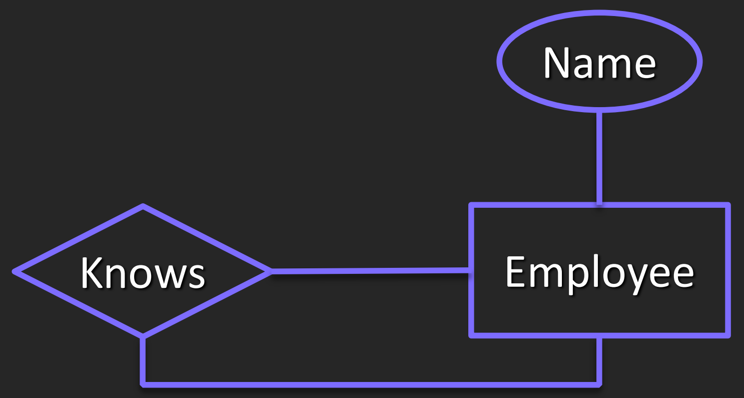
Objective

Which library users have books currently lent which are over-due?

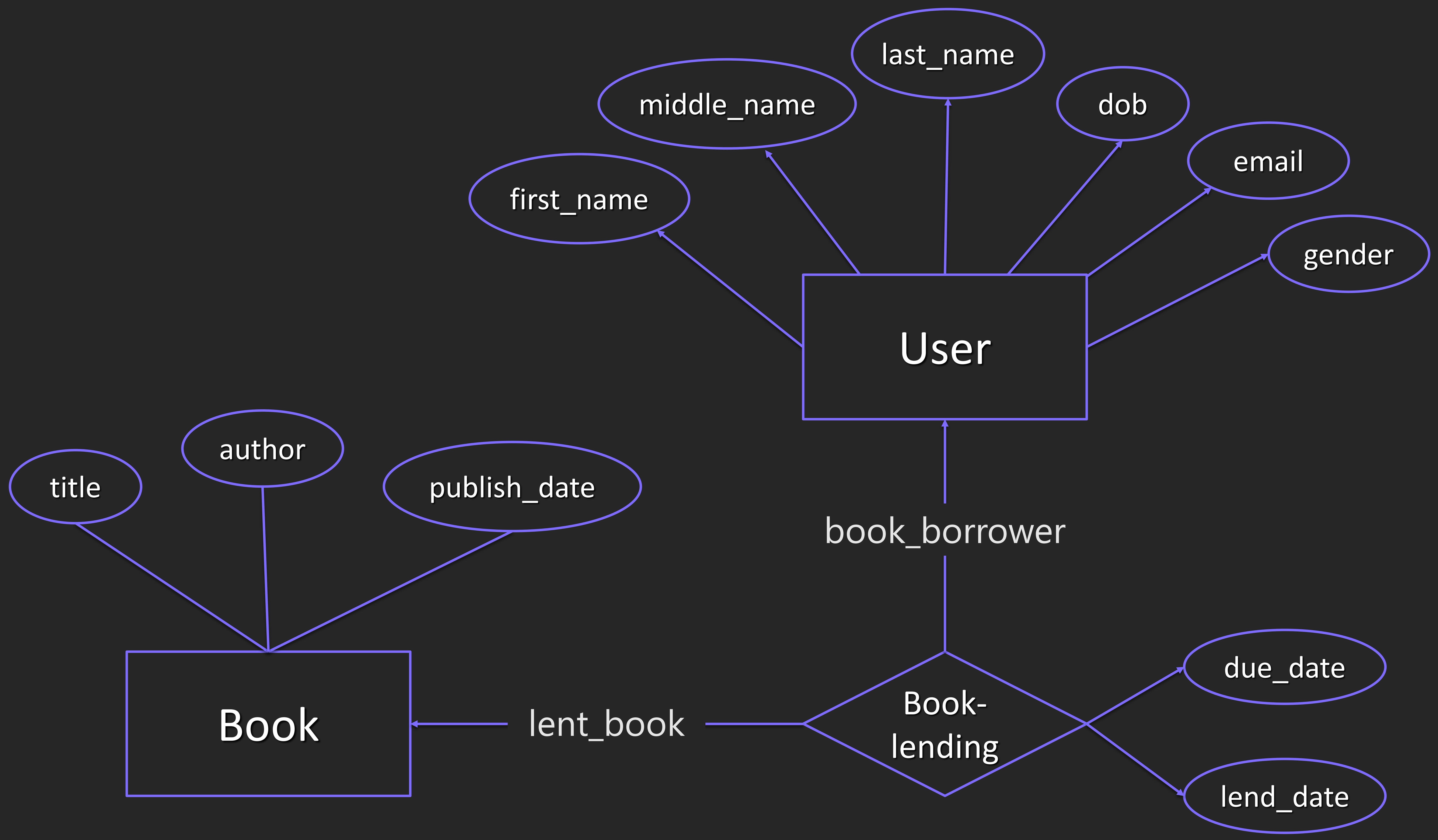
Data Available

Book
- Author
- Title
- Publish date
- Lend date
- Due date

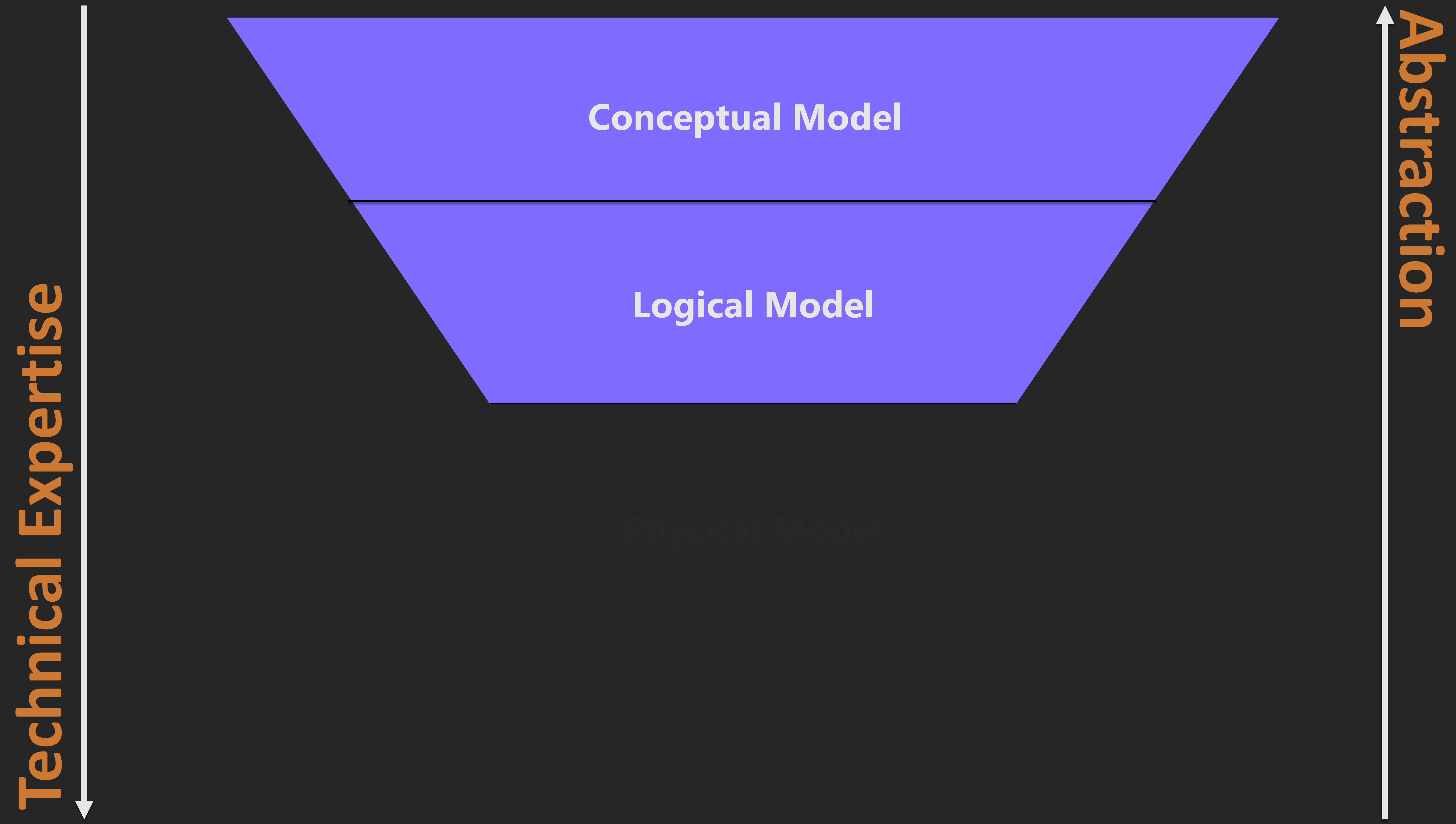
User
- First/middle/last name
- DOB
- Email
- Gender



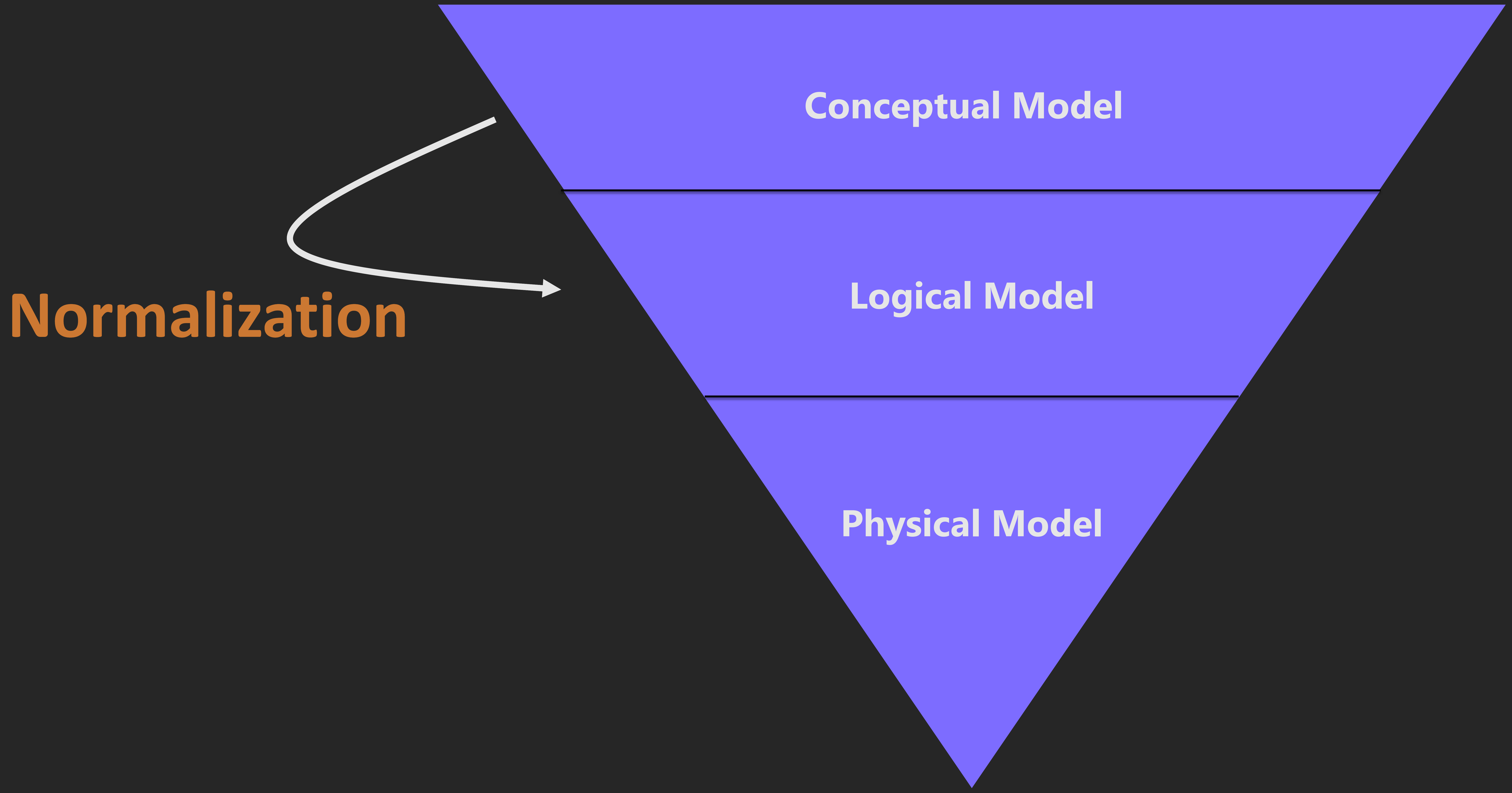
Conceptual Model



Data Modeling



Data Modeling



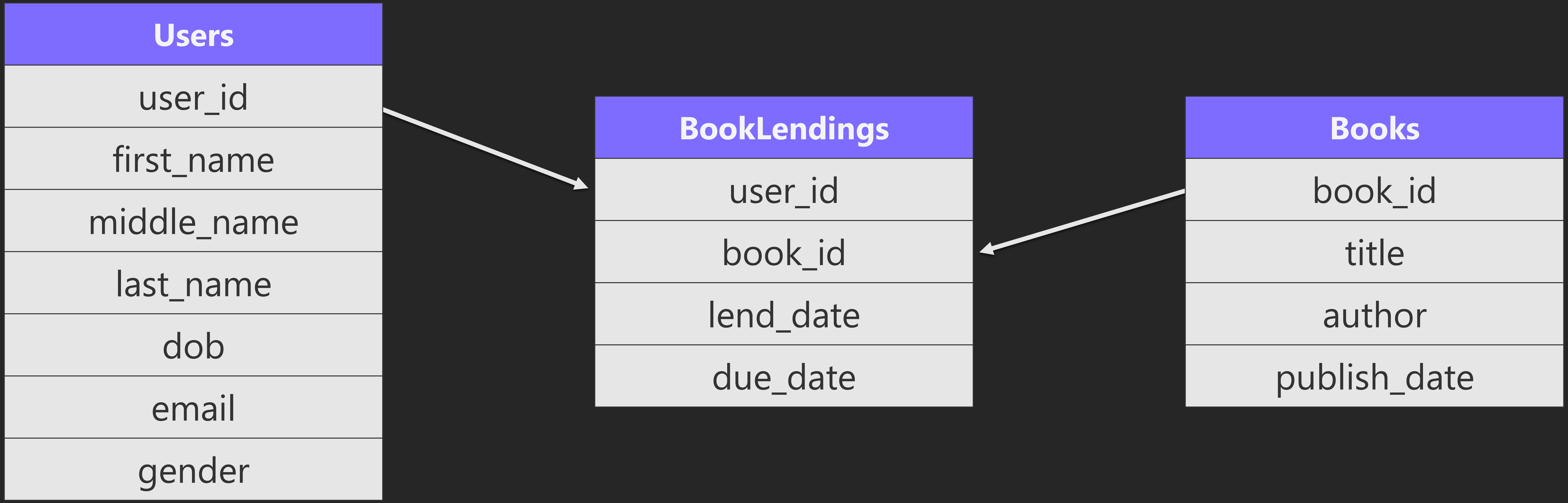
Conceptual Model

Logical Model

Physical Model

Normalization

Data Modeling

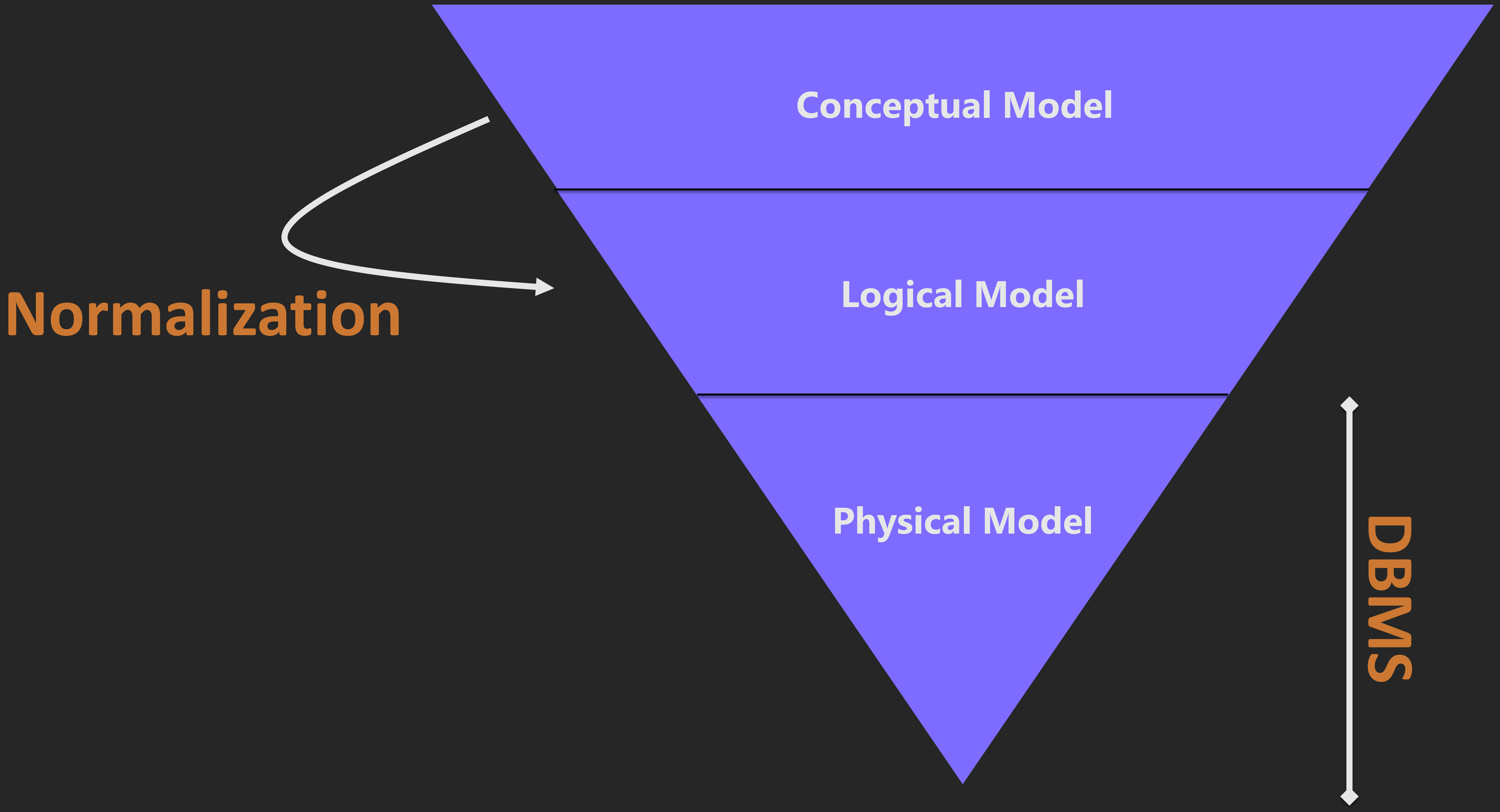


Users
user_id
first_name
middle_name
last_name
dob
email
gender

BookLendings
user_id
book_id
lend_date
due_date

Books
book_id
title
author
publish_date

Data Modeling

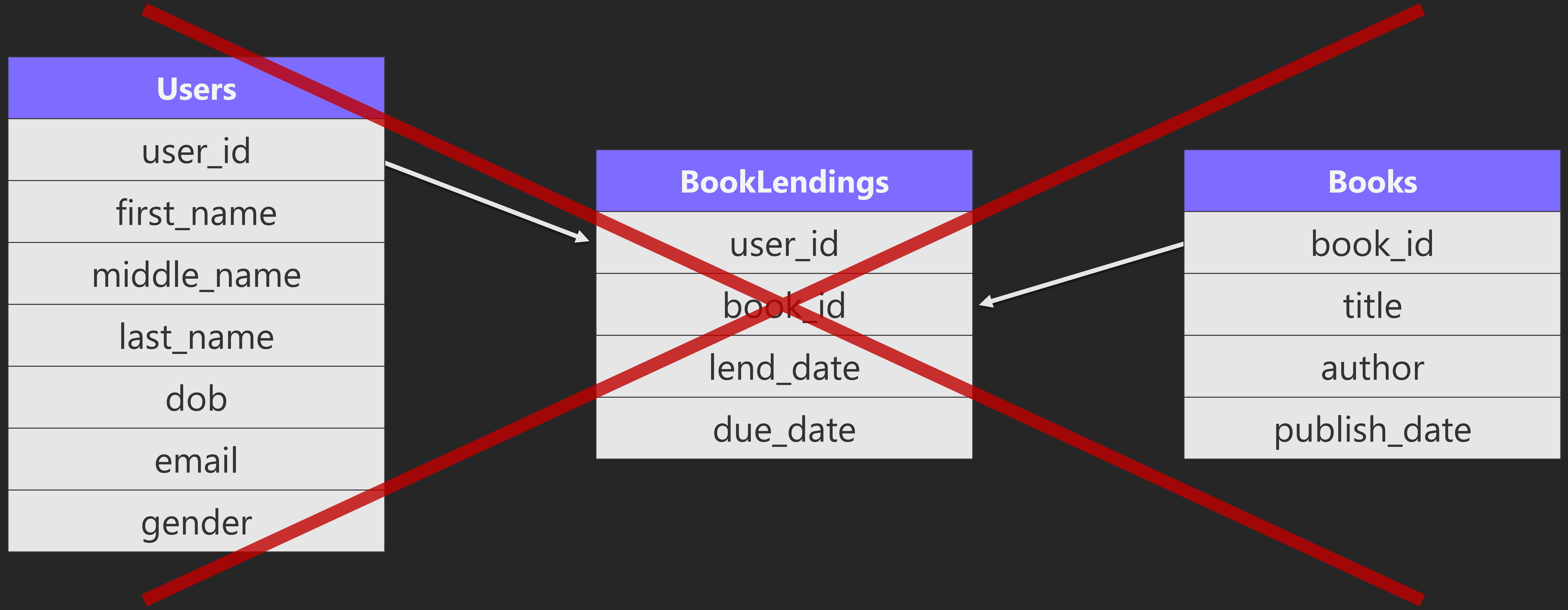


Introducing Grakn

GRAKN.AI

Knowledge Representation System

Data Modeling

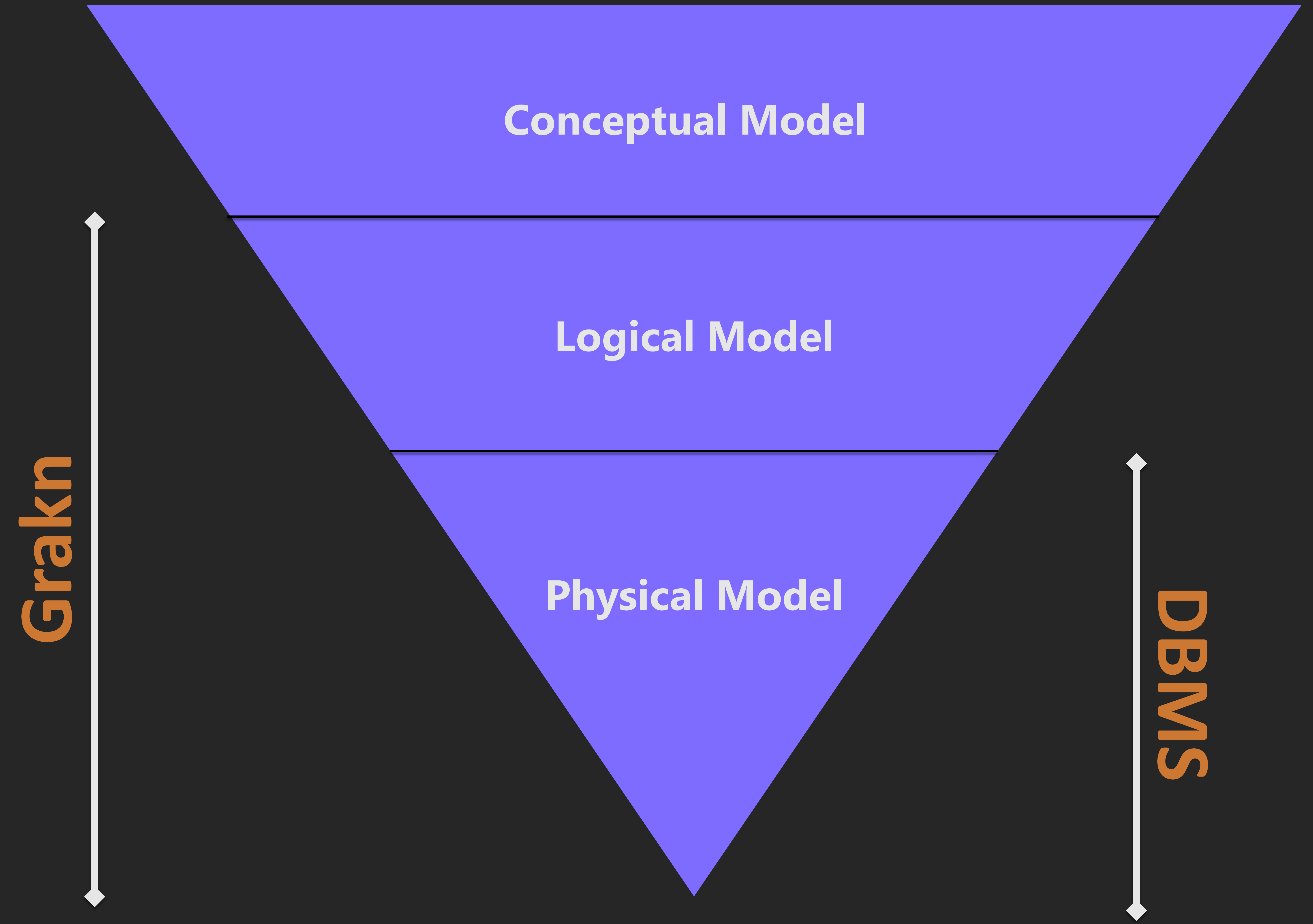


Users
user_id
first_name
middle_name
last_name
dob
email
gender

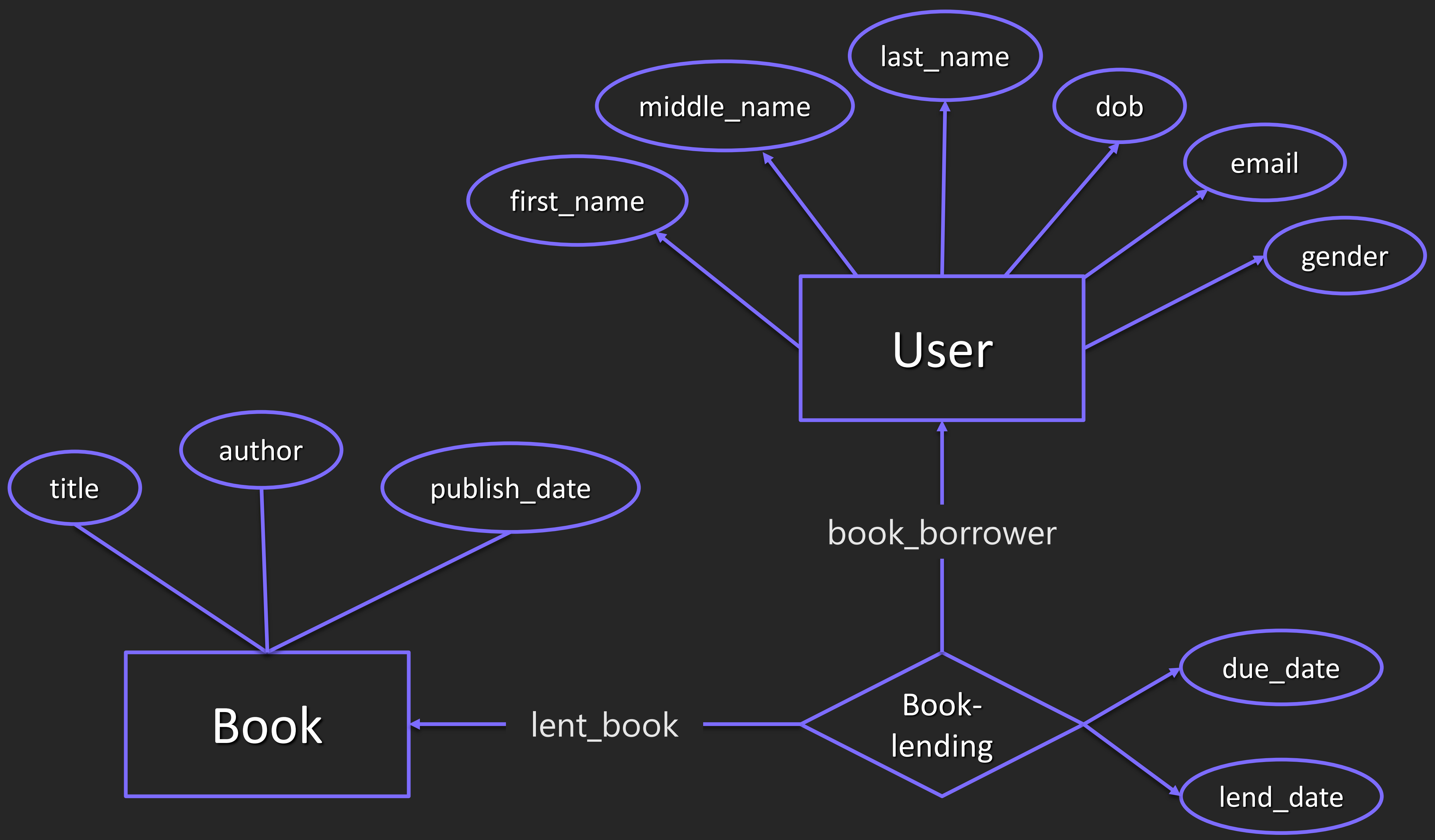
BookLendings
user_id
book_id
lend_date
due_date

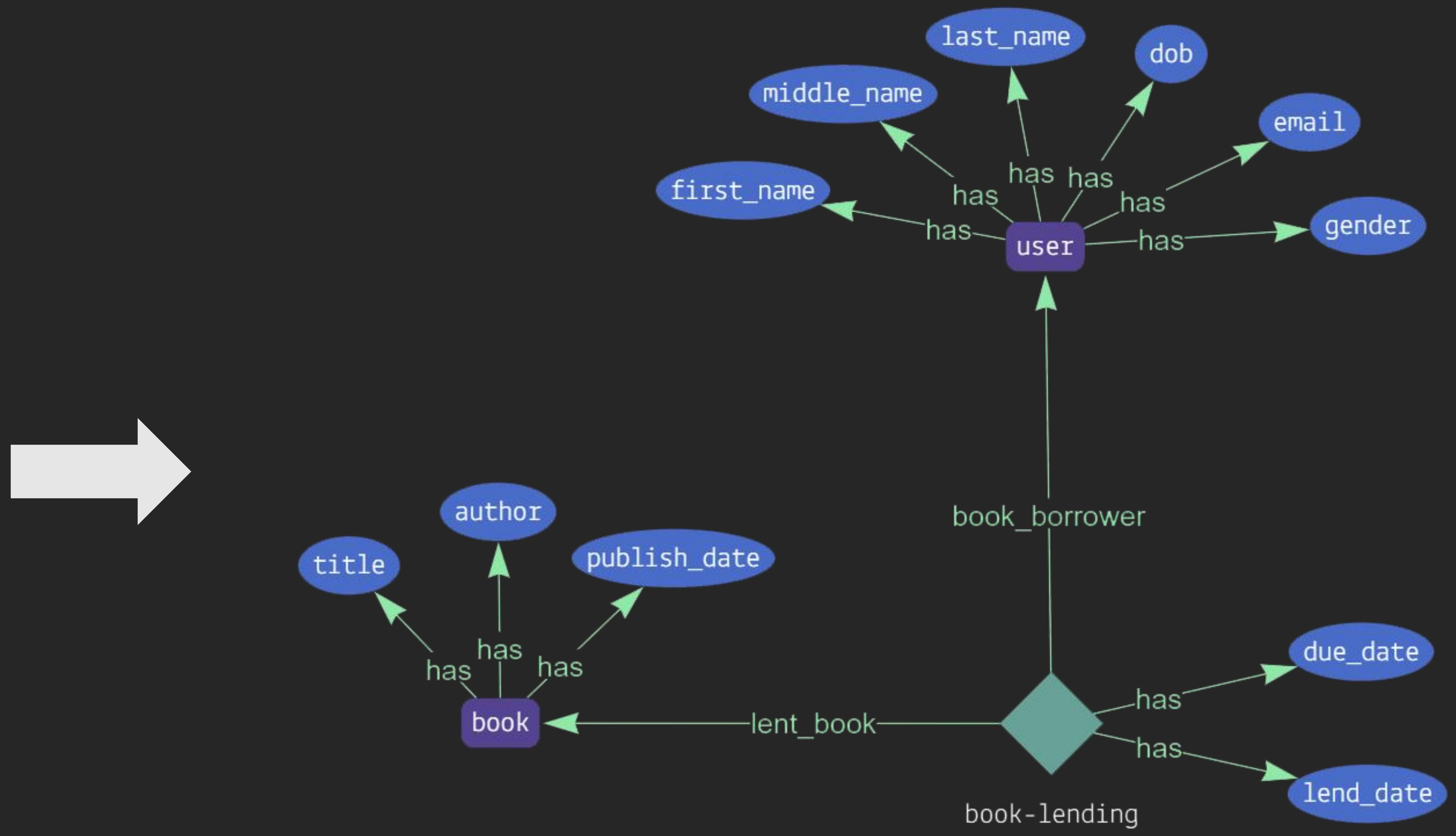
Books
book_id
title
author
publish_date

Data Modeling



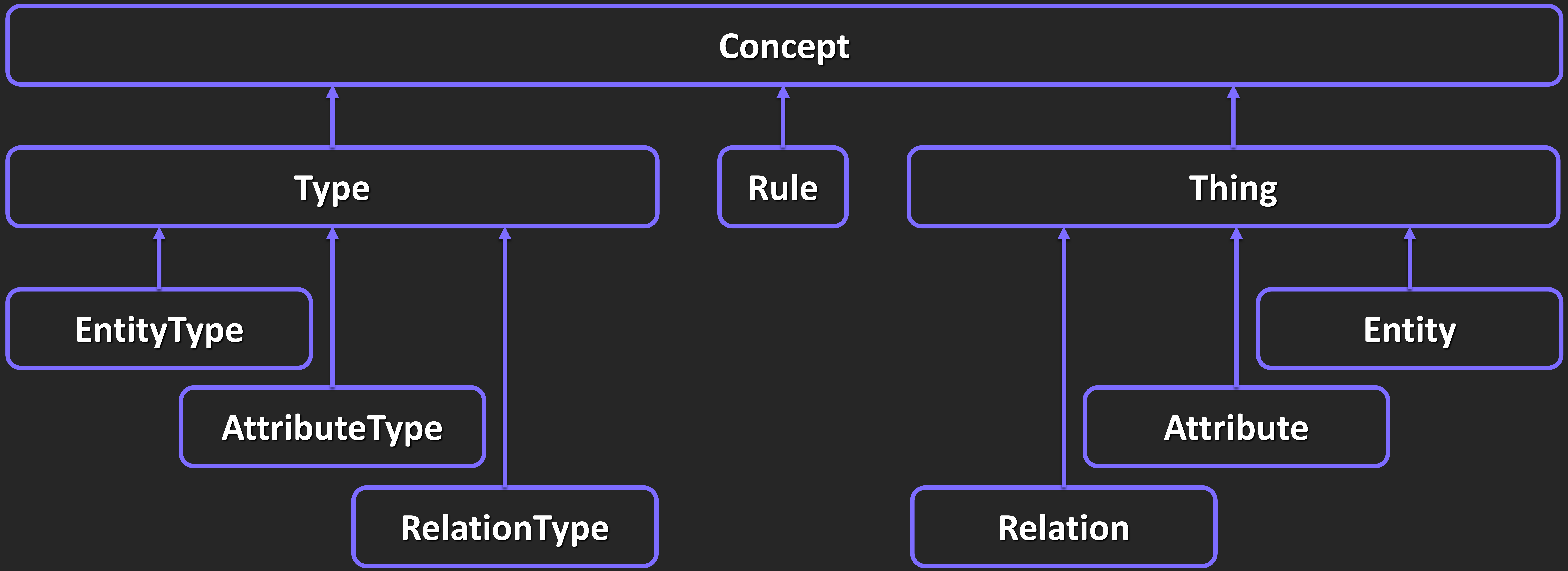
Data Modeling





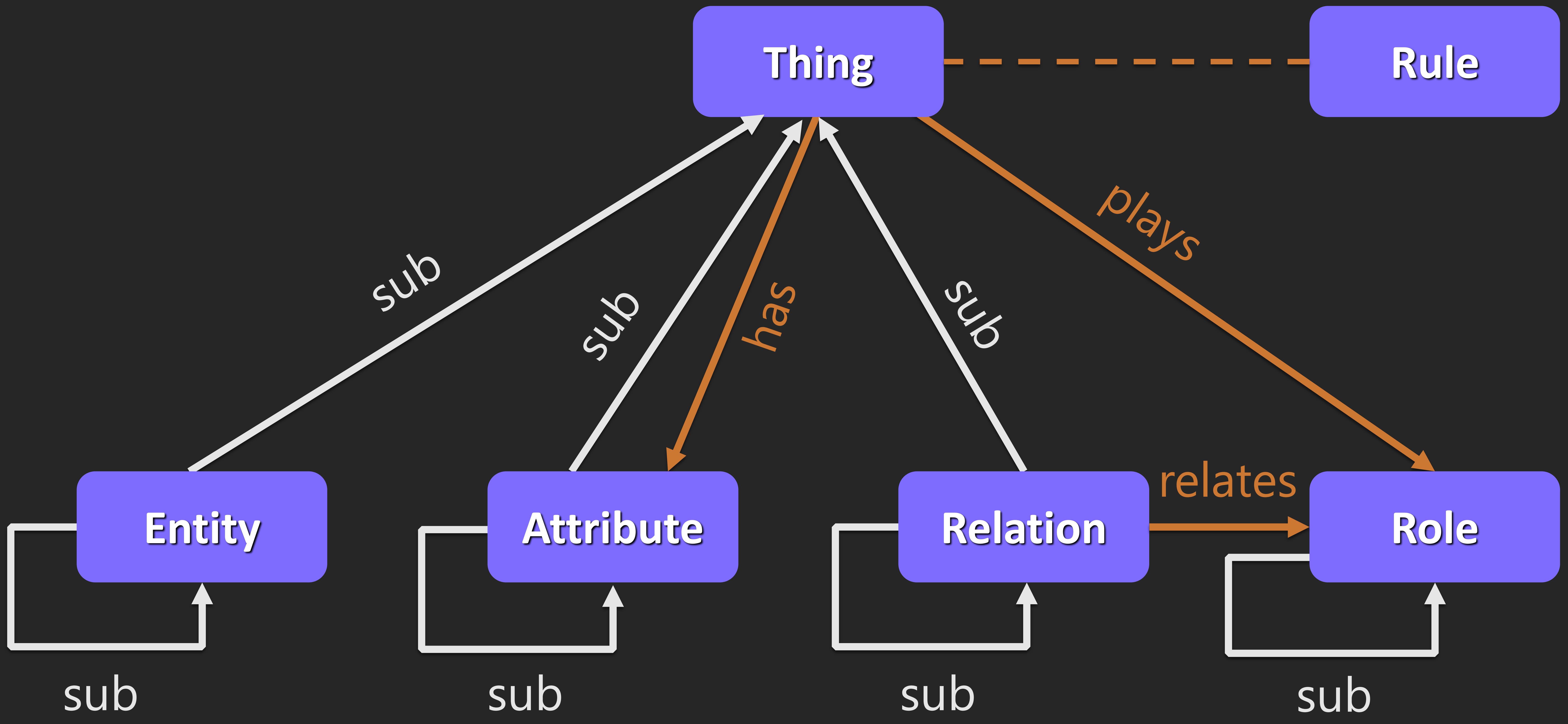
Grakn Ontology

A highly expressive and intelligent
type system for your complex data

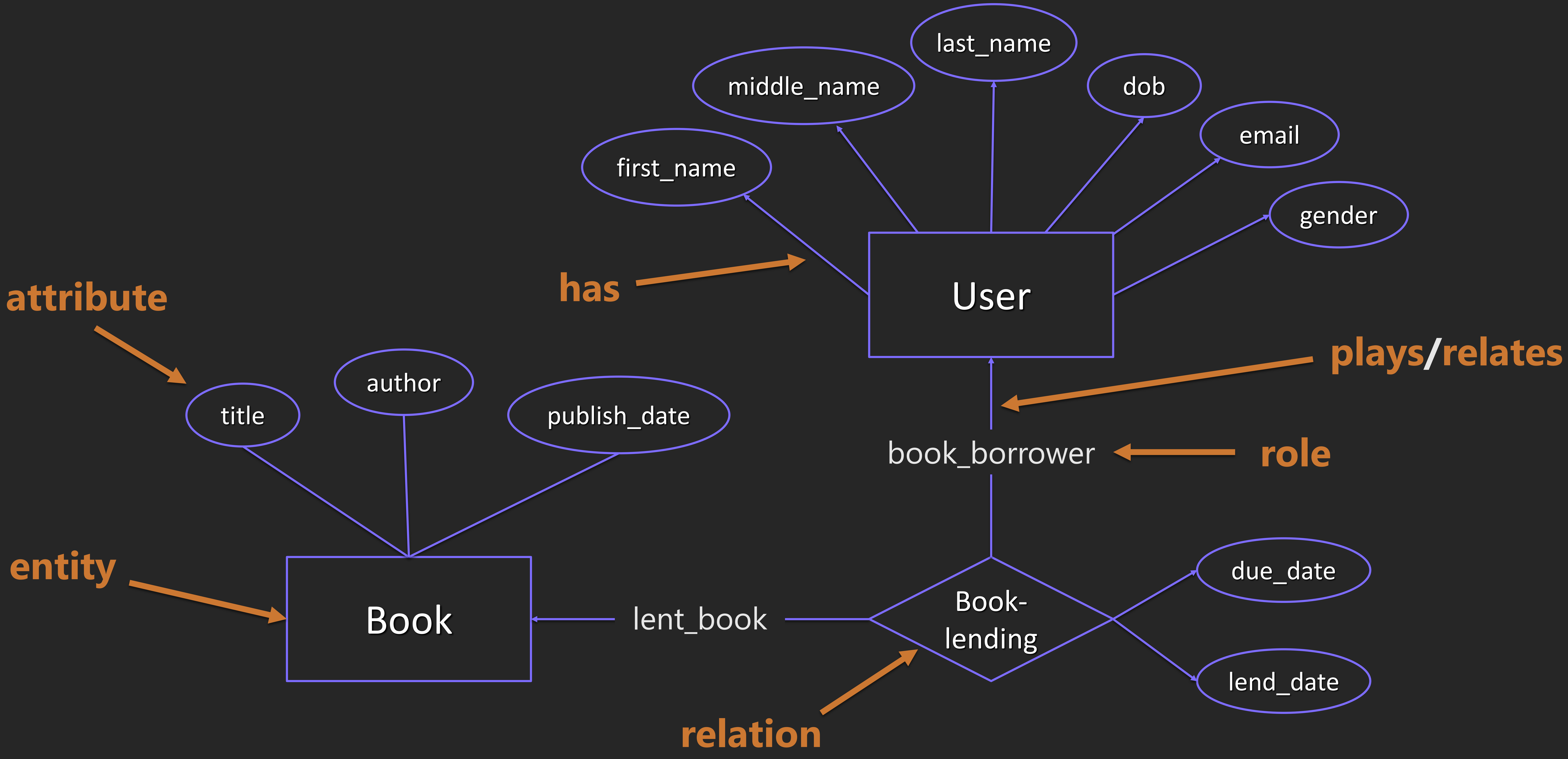


Knowledge Model

A schema which can represent:
type hierarchies, hyper-relations and rules

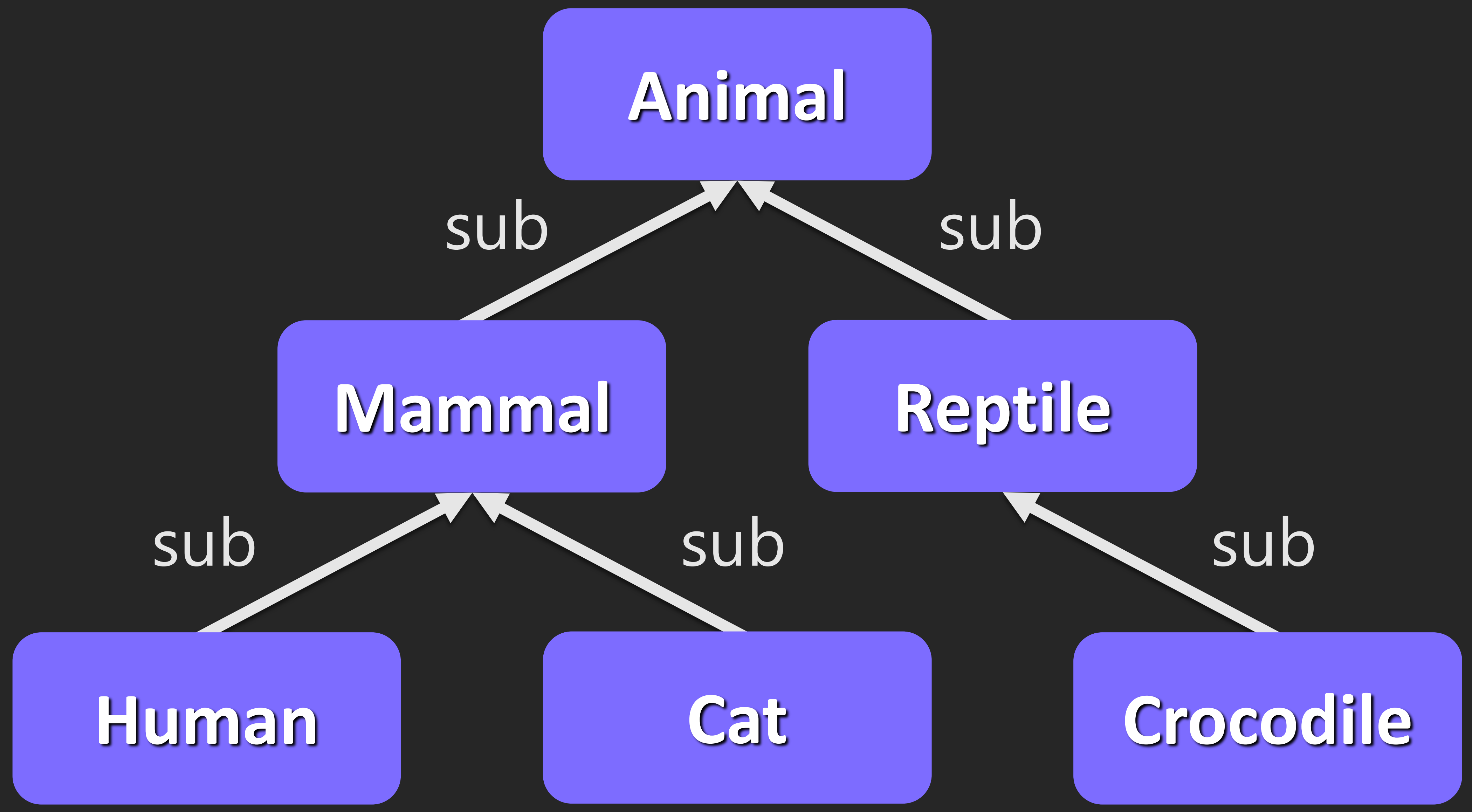


Knowledge Model

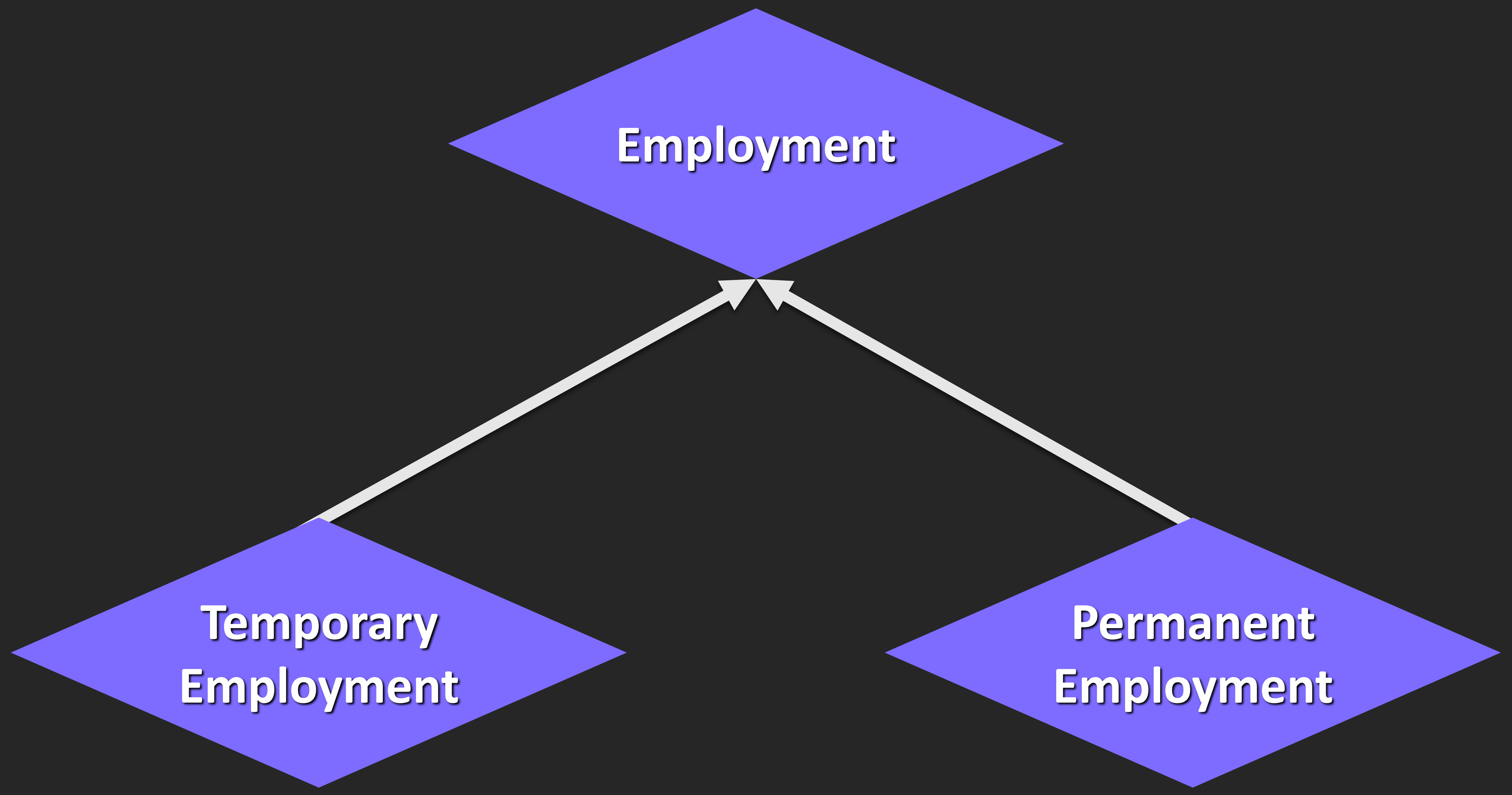


Grakn Modeling – Hierarchies

Entity Type Hierarchies

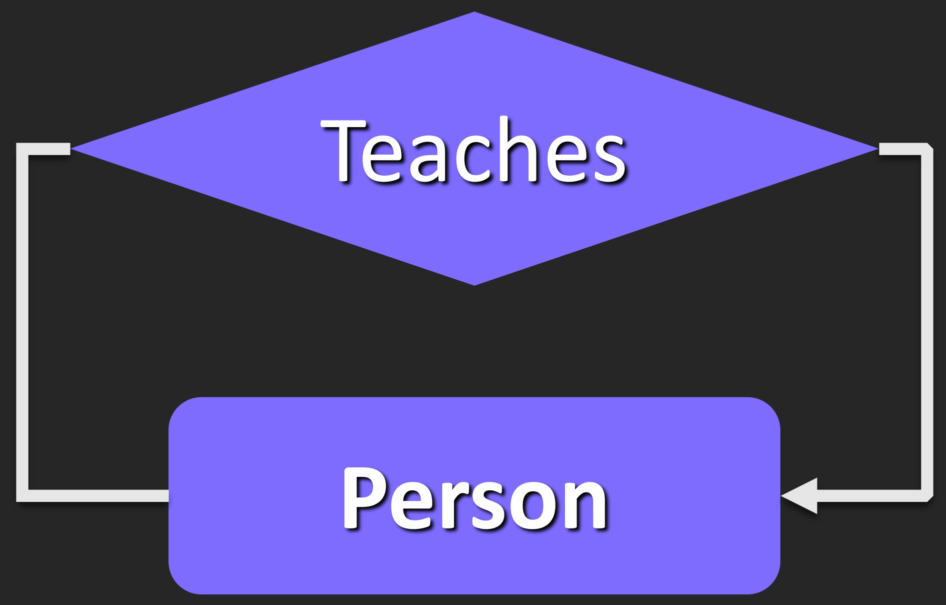


Relation Type Hierarchies

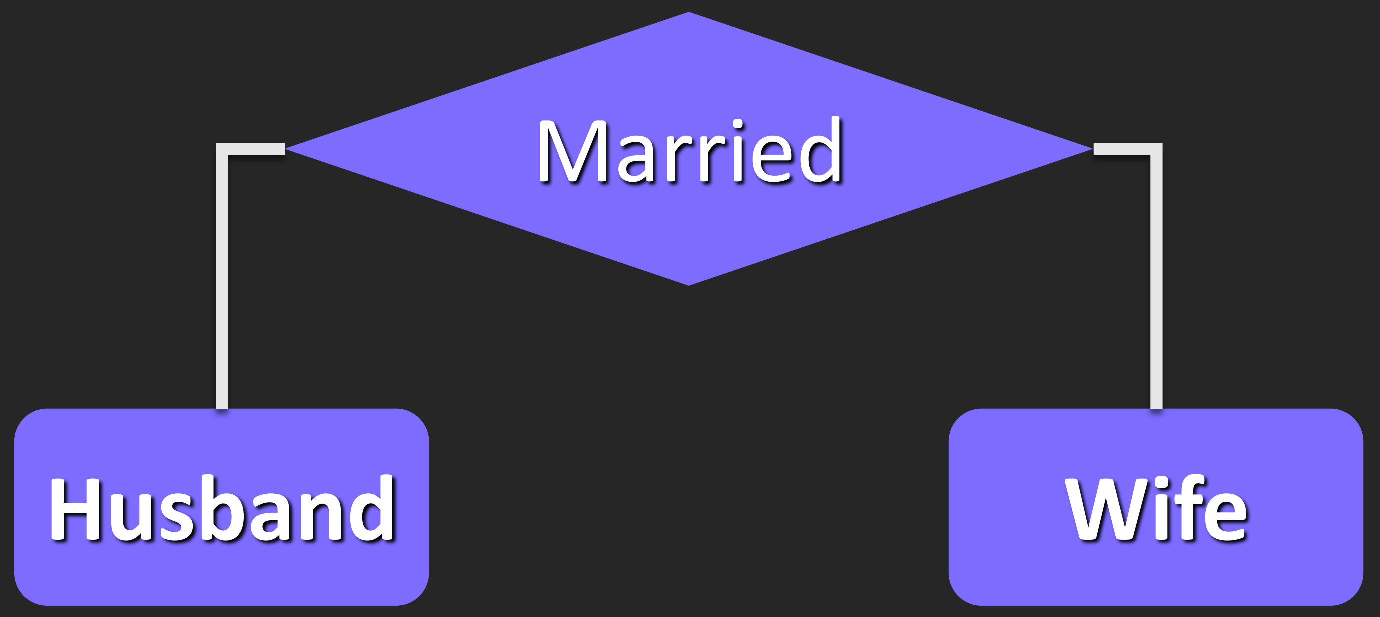


Grakn Modeling – Relations

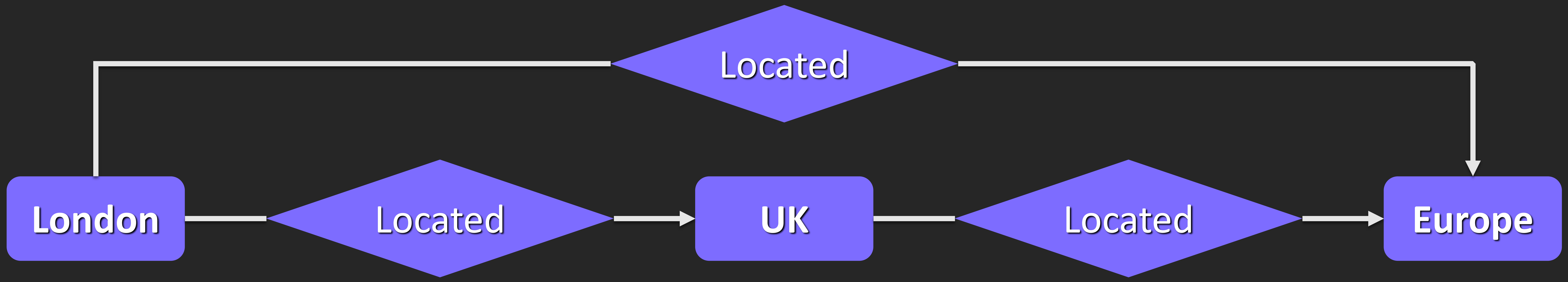
Reflexive Relations



Symmetric Relations

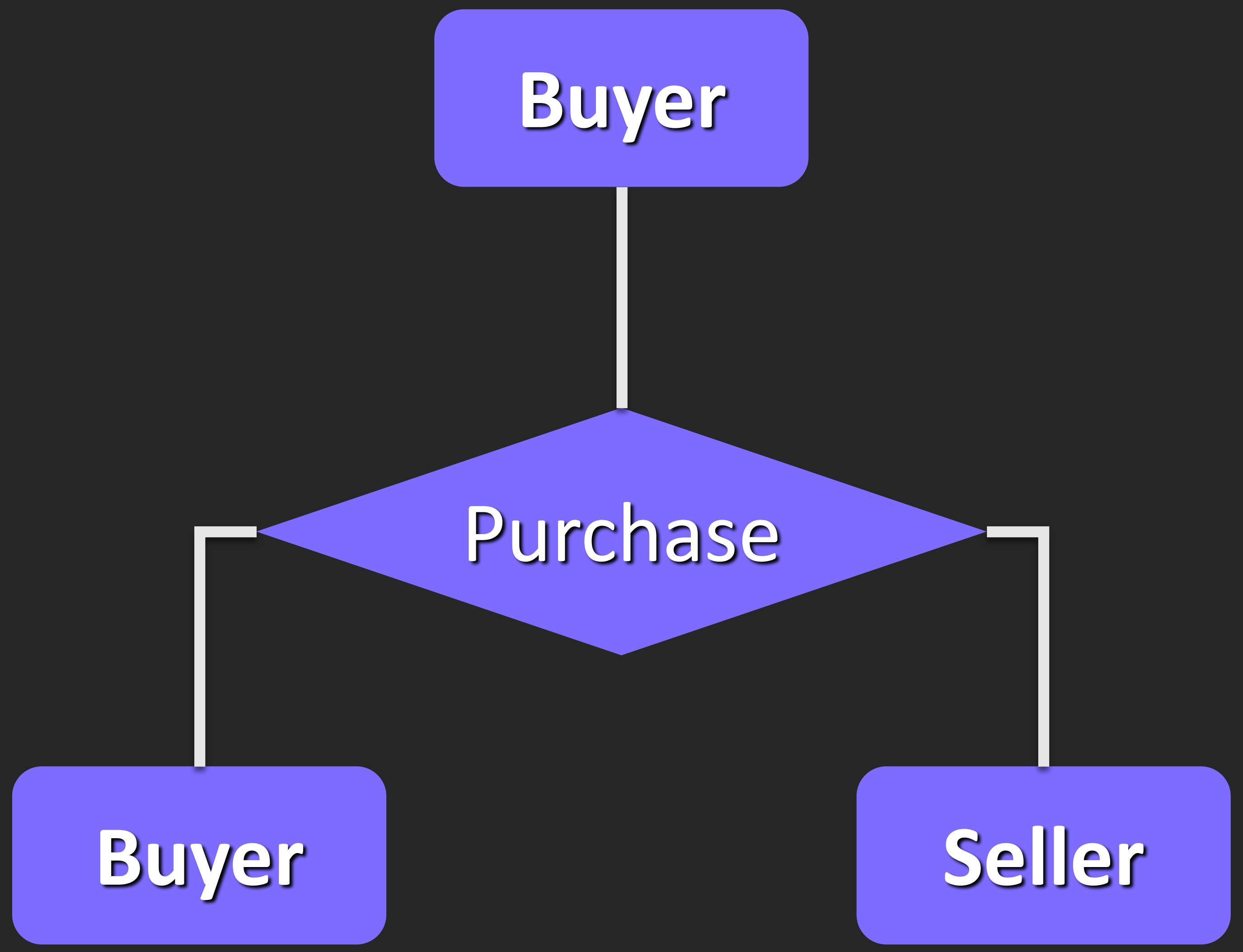


Transitive Relations

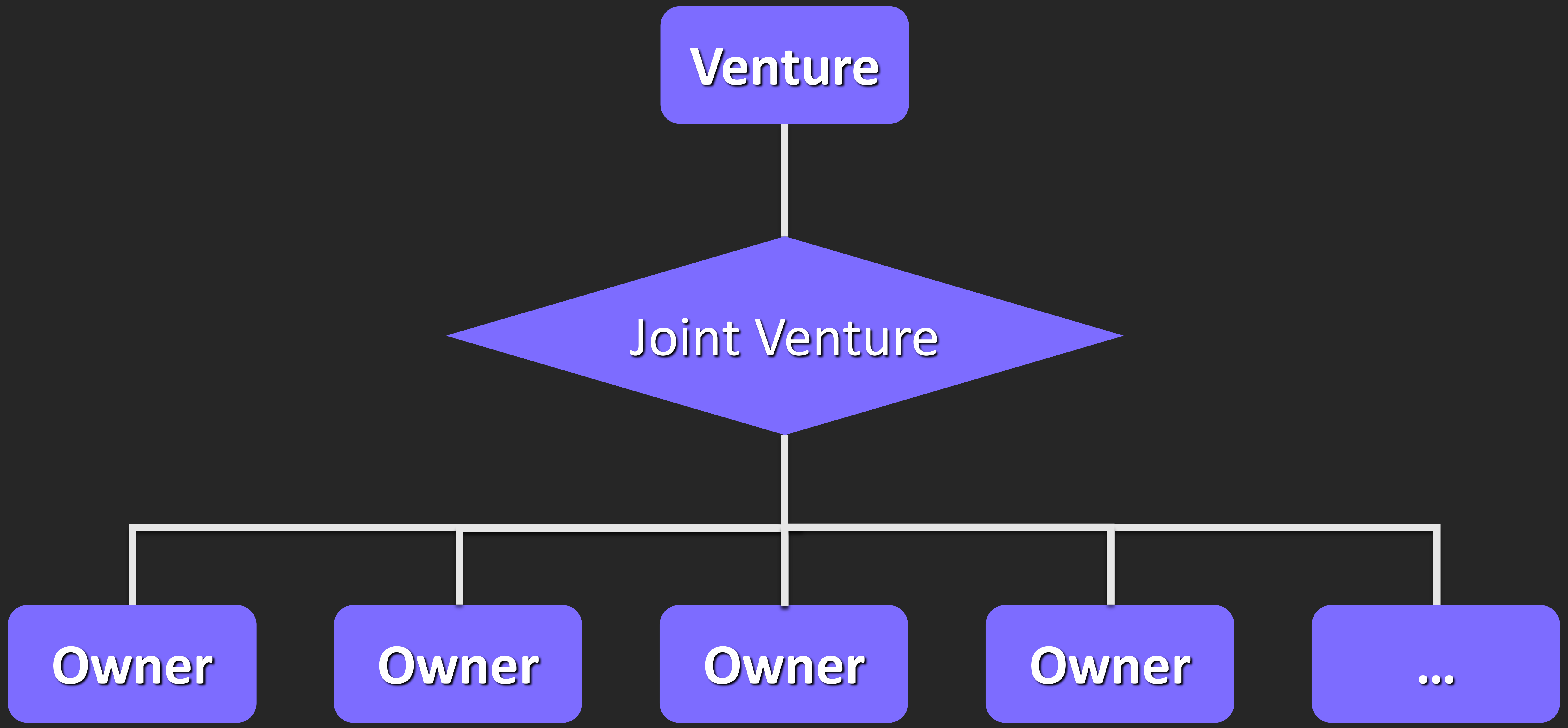


Grakn Modeling – Relations

Ternary Relations

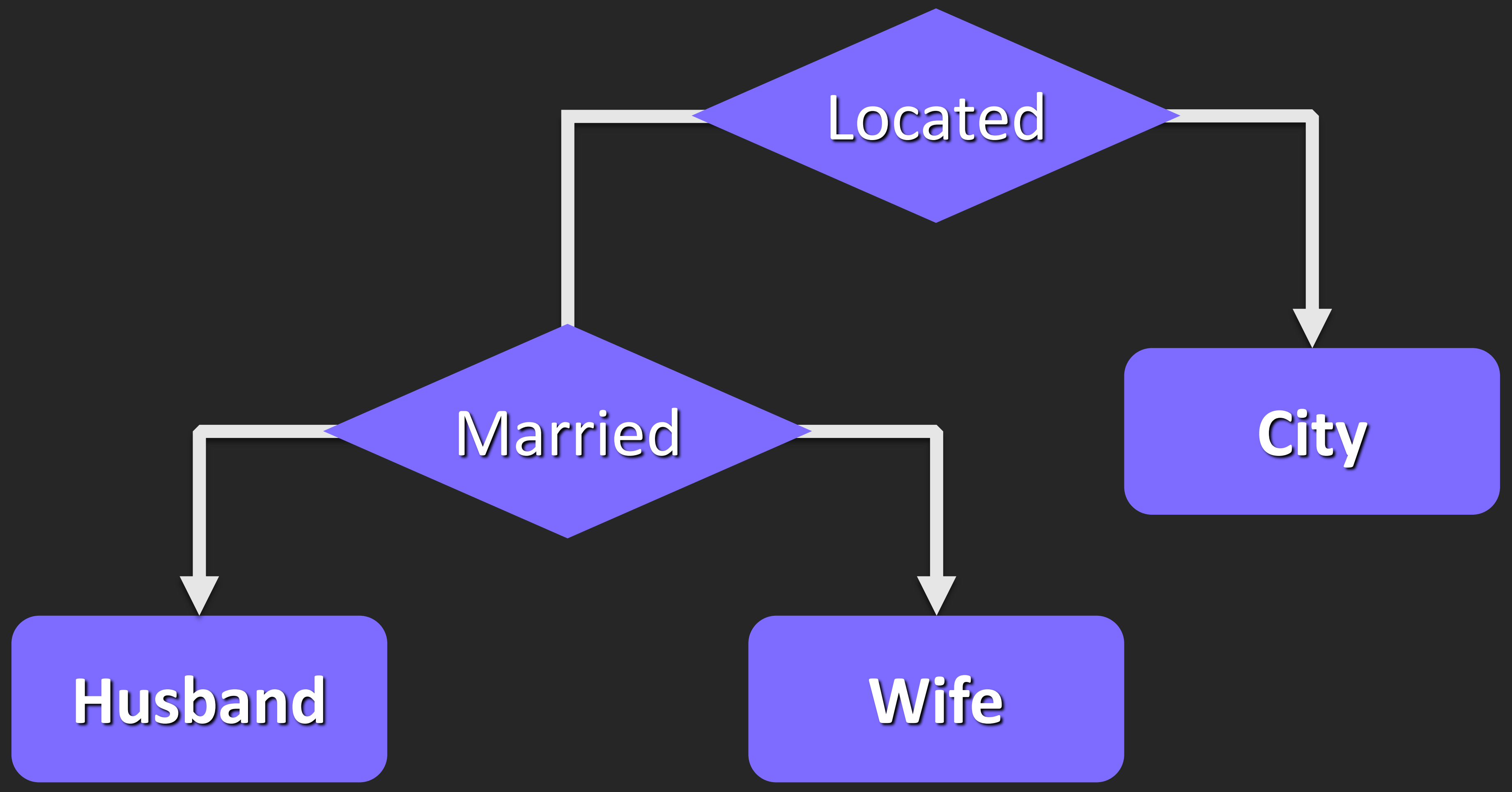


Infinitary Relations

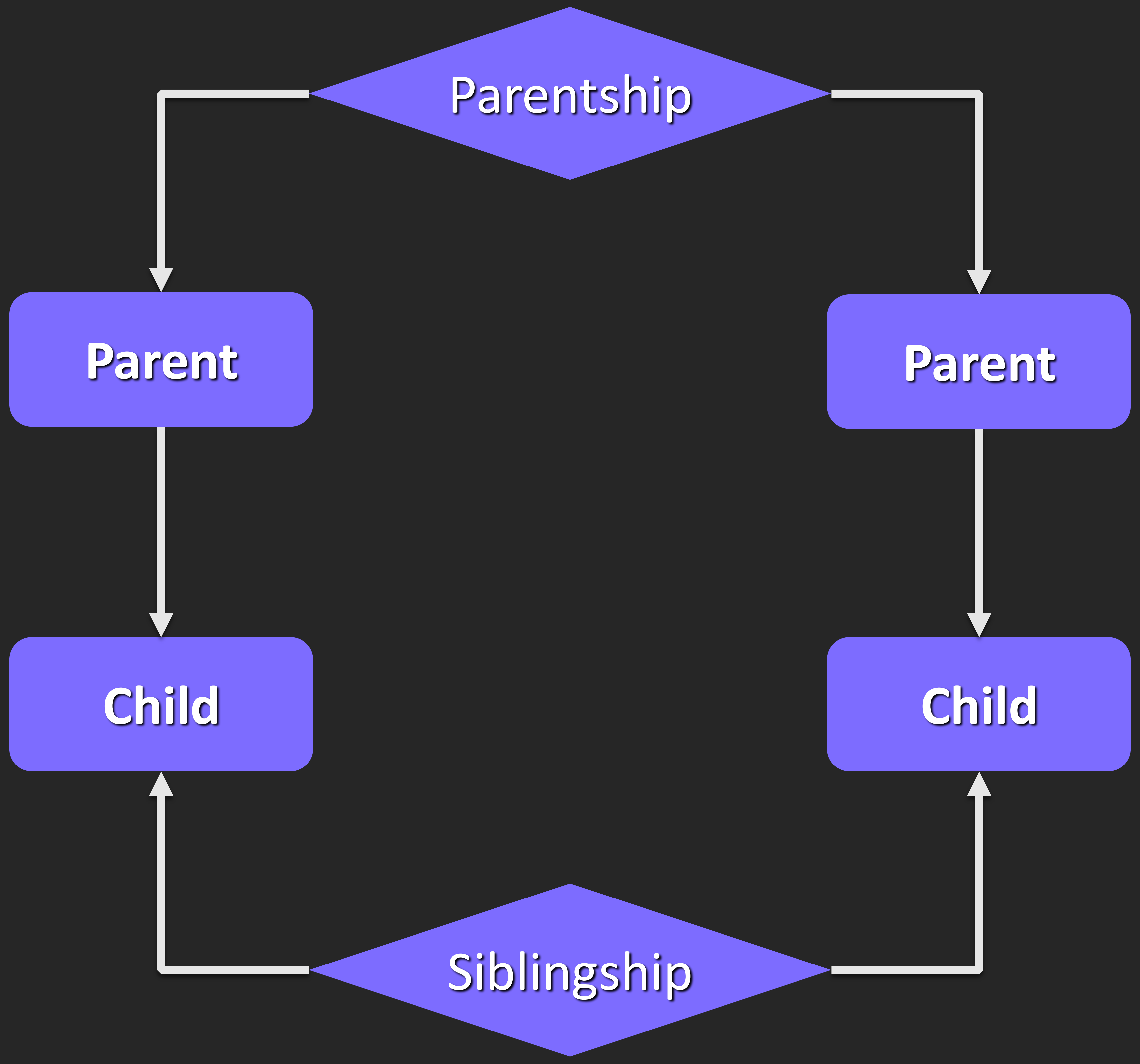


Grakn Modeling – Relations

Nested Relations



Equivalent Relations



Grakn Modeling – Tips

Model incrementally

- Start with questions you want answered now

Be as specific as possible

- father > parent

Relations are first-class citizens

- No foreign keys!

Generally: Tables are entities, rows are instances, columns are attributes

- Associative/junction tables are relations

Unconnected data is hard to reach

- Relations provide easy access to connected data

No such thing as nulls

- The absence of value indicates the lack of value

Prefer ingesting non-changing data (facts) over syncing volatile data

- Represent new data (new facts) with additional values

Introducing Graql

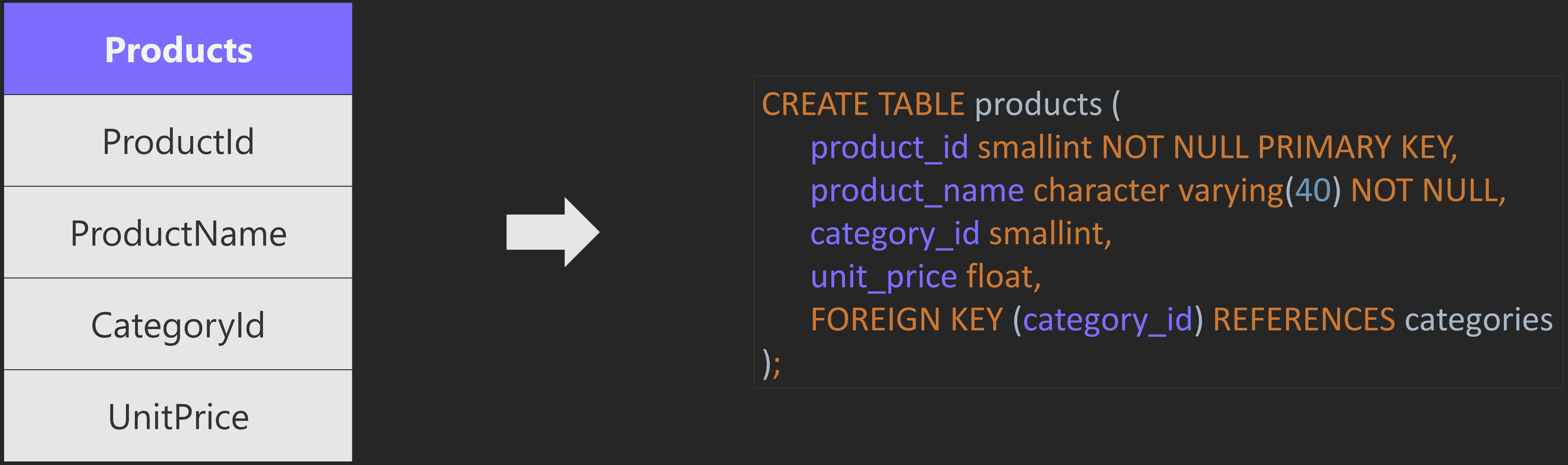
Commands

CREATE/ALTER TABLE	->	define
SELECT+WHERE	->	match+get
INSERT INTO	->	insert
DELETE	->	delete
DROP TABLE	->	undefine
UPDATE	->	match+delete+insert

Data Types

NUMERIC	->	long
DECIMAL	->	double
TEXT	->	string
BOOLEN	->	boolean
DATE	->	date

Defining Schemas

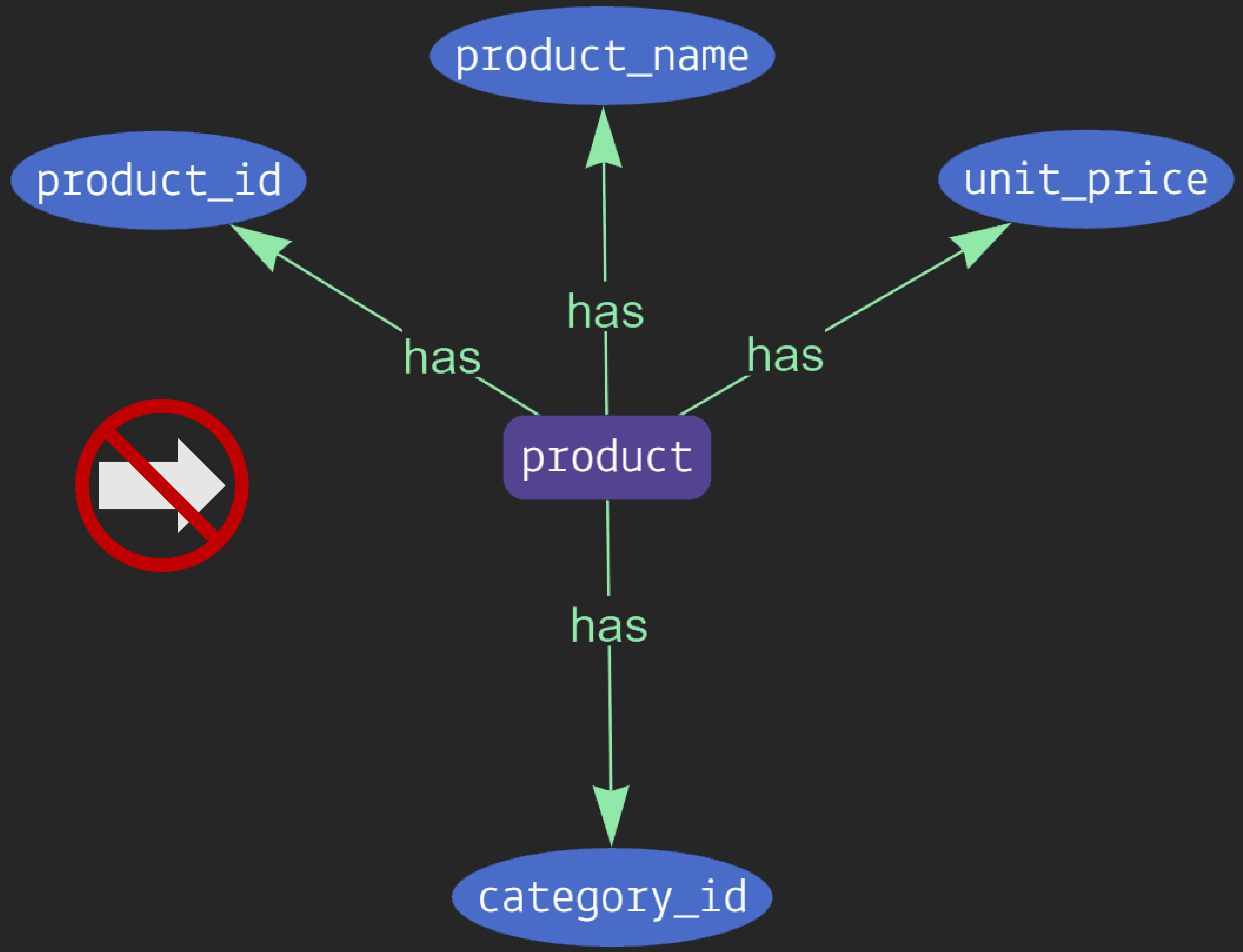


Products
ProductId
ProductName
CategoryId
UnitPrice

```sql
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```

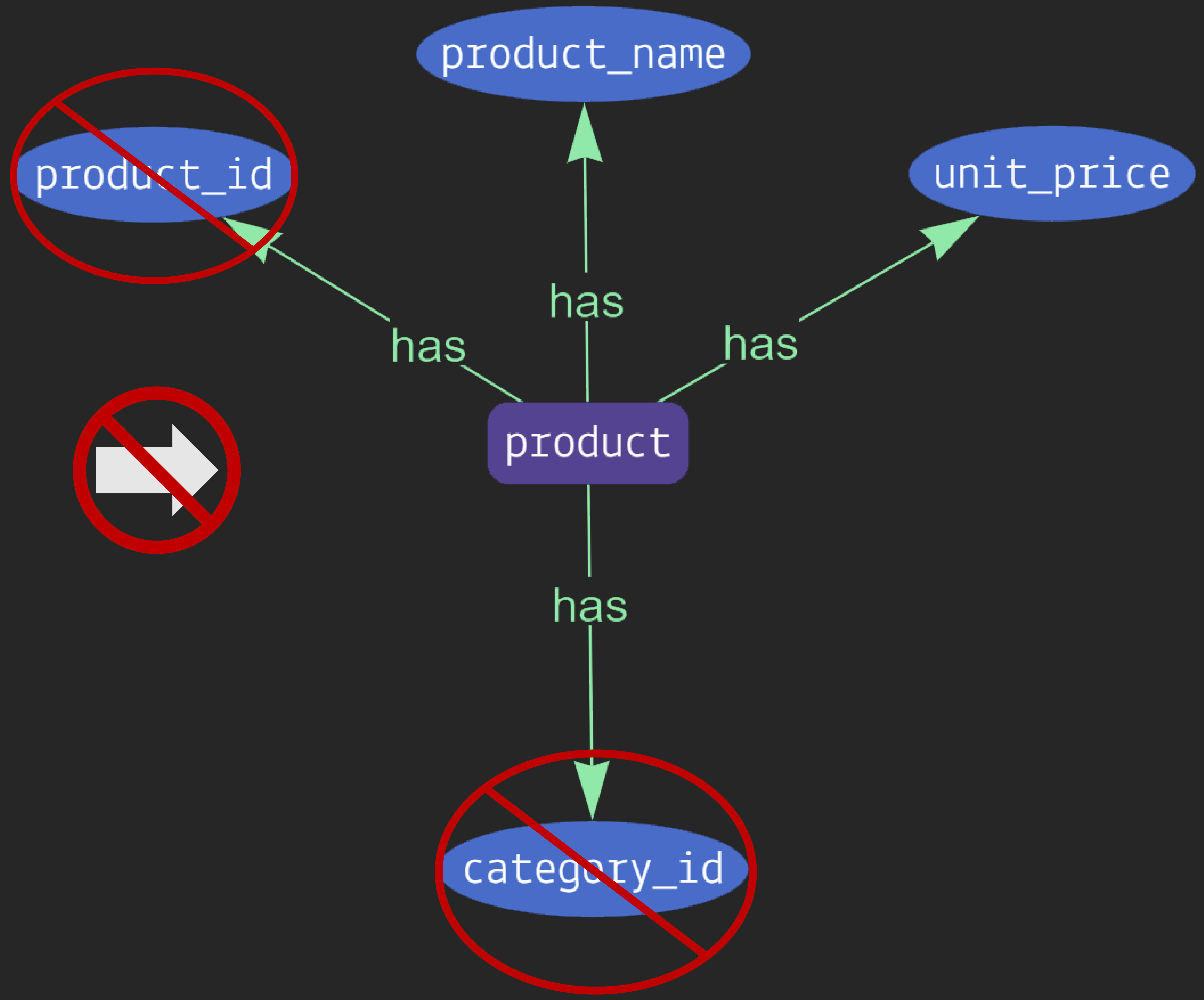
Defining Schemas

```
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```



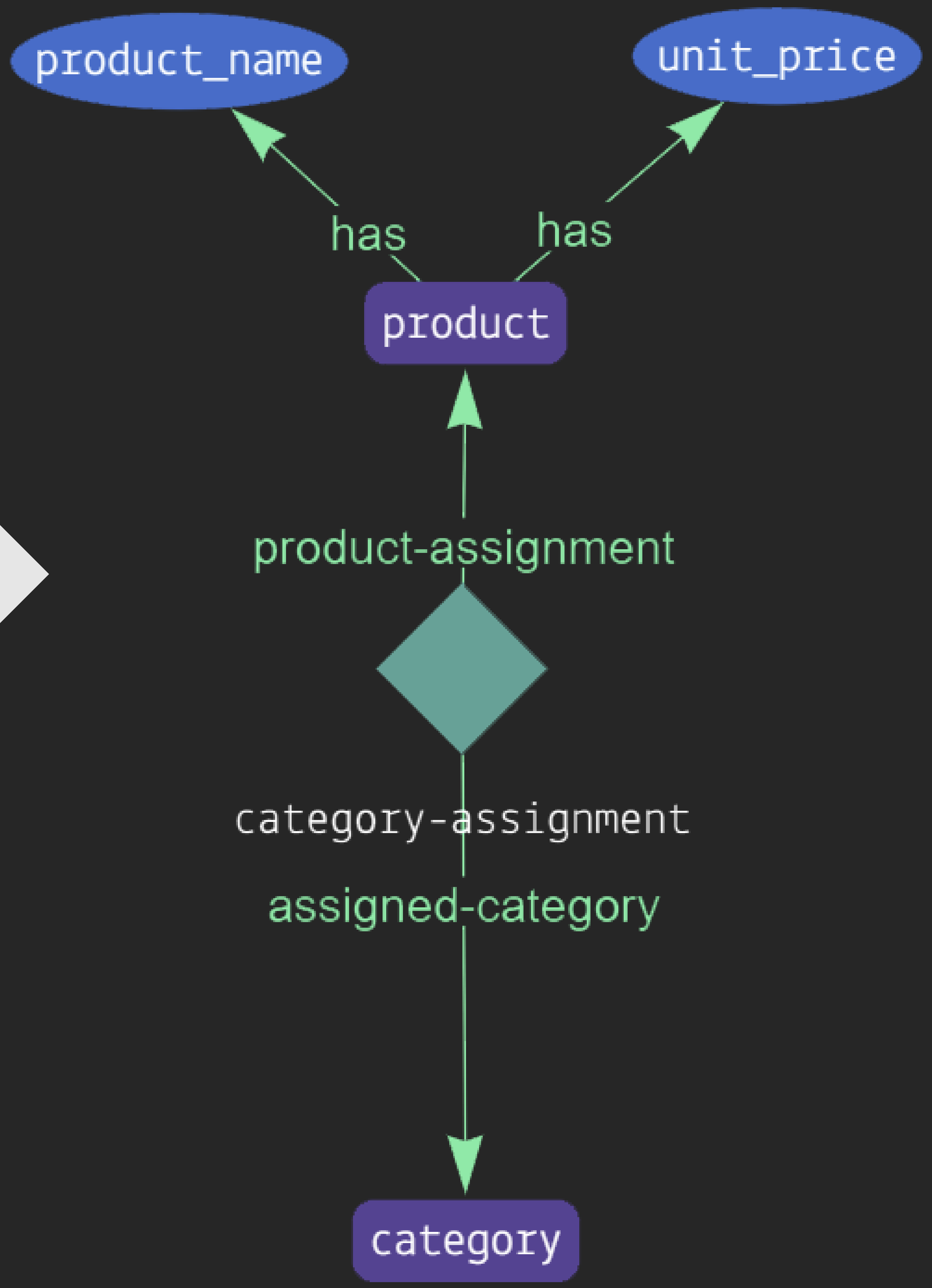
Defining Schemas

```
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```



Defining Schemas

```
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```



SQL

```
CREATE TABLE products (
   product_id smallint NOT NULL PRIMARY KEY,
   product_name character varying(40) NOT NULL,
   category_id smallint,
   unit_price float,
   FOREIGN KEY (category_id) REFERENCES categories
);

CREATE TABLE categories (
   category_id smallint NOT NULL PRIMARY KEY,
   category_name character varying(40)
);
```

Graql

```
define
product sub entity,
   key product_name,
   has unit_price,
   plays product-assignment;
category sub entity,
   has category_name,
   plays assigned-category;

product_name sub attribute, datatype string;
unit_price sub attribute, datatype double;
category_name sub attribute, datatype string;

category-assignment sub relation,
   relates assigned-category,
   relates product-assignment;
```

Defining Schemas

SQL

```
CREATE TABLE albums (
   album_id SMALLINT NOT NULL PRIMARY KEY,
   album_name CHARACTER VARYING(40) NOT NULL,
   release_date DATE NOT NULL,
   artist_id SMALLINT NOT NULL,
   FOREIGN KEY (artist_id) REFERENCES artists
);

CREATE TABLE artists (
   artist_id smallint NOT NULL PRIMARY KEY,
   artist_name character varying(40)
);
```

Graql

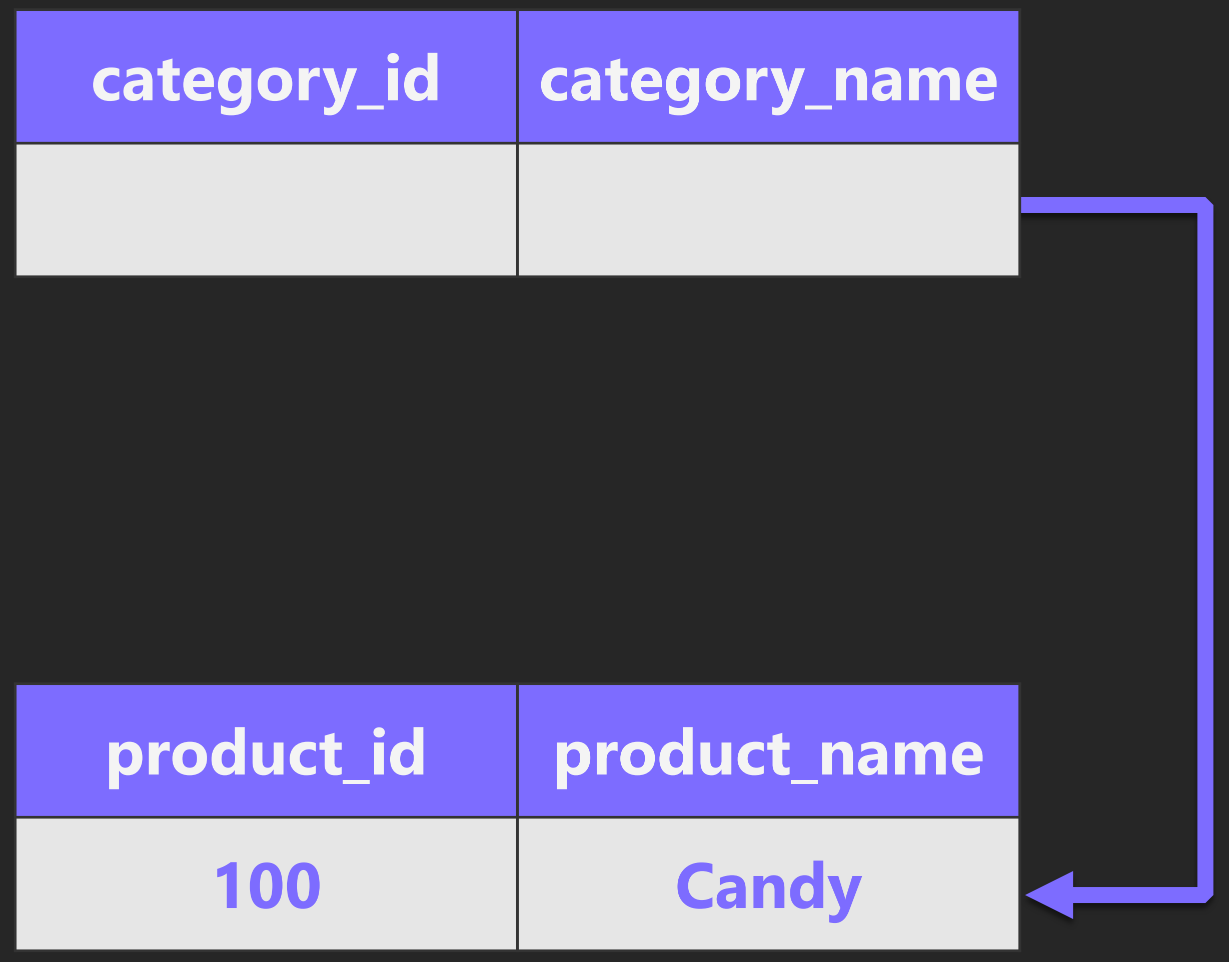
★ Fill in the Blanks ★

```
______
album sub  ______,
   key album_name,
   has release_date,
   ______ released_album;
artist sub entity,
   ___ artist_name,
   plays releasing_artist;

album-release sub relation,
   relates released_album,
   relates releasing_artist;

album_name sub  _______, datatype string;
release_date sub attribute, datatype date;
artist_name sub attribute, datatype _______;
```

★ Fill in the Blanks ★

Defining Schemas

SQL

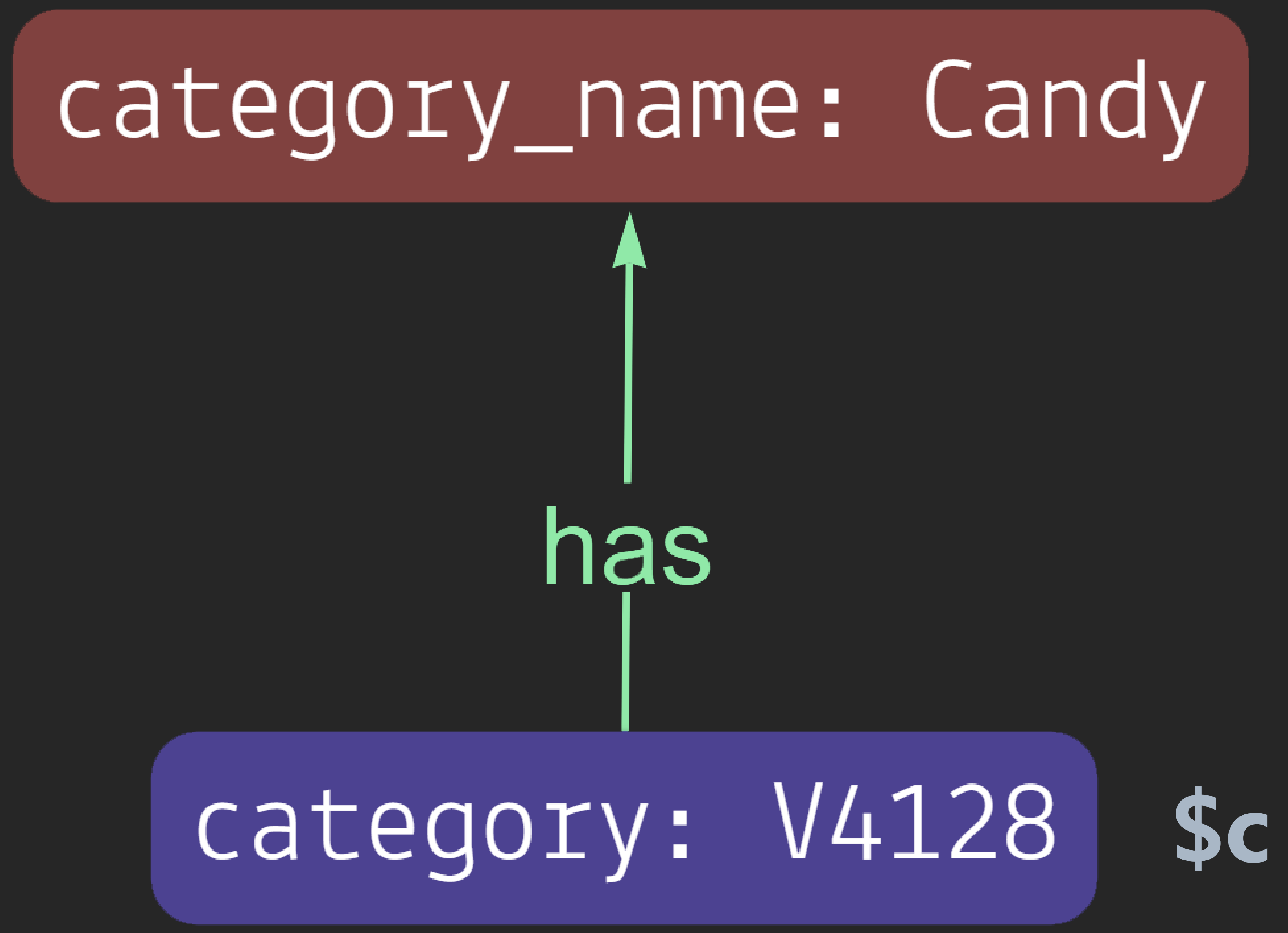


category_id	category_name

product_id	product_name
100	Candy

```
INSERT INTO categories (category_id, category_name)
VALUES (100, 'Candy');
```

Graql



```
insert $c isa category,
  has category_name "Candy";
```

Writing Data – Insert

SQL

```
INSERT INTO artists (artist_id, artist_name)
VALUES (1, 'Michael Jackson');
```

Graql

★ Fill in the Blanks ★

```
_______ $a isa _______,
  has ____________ "Michael Jackson";
```

★ Fill in the Blanks ★

Writing Data – Insert

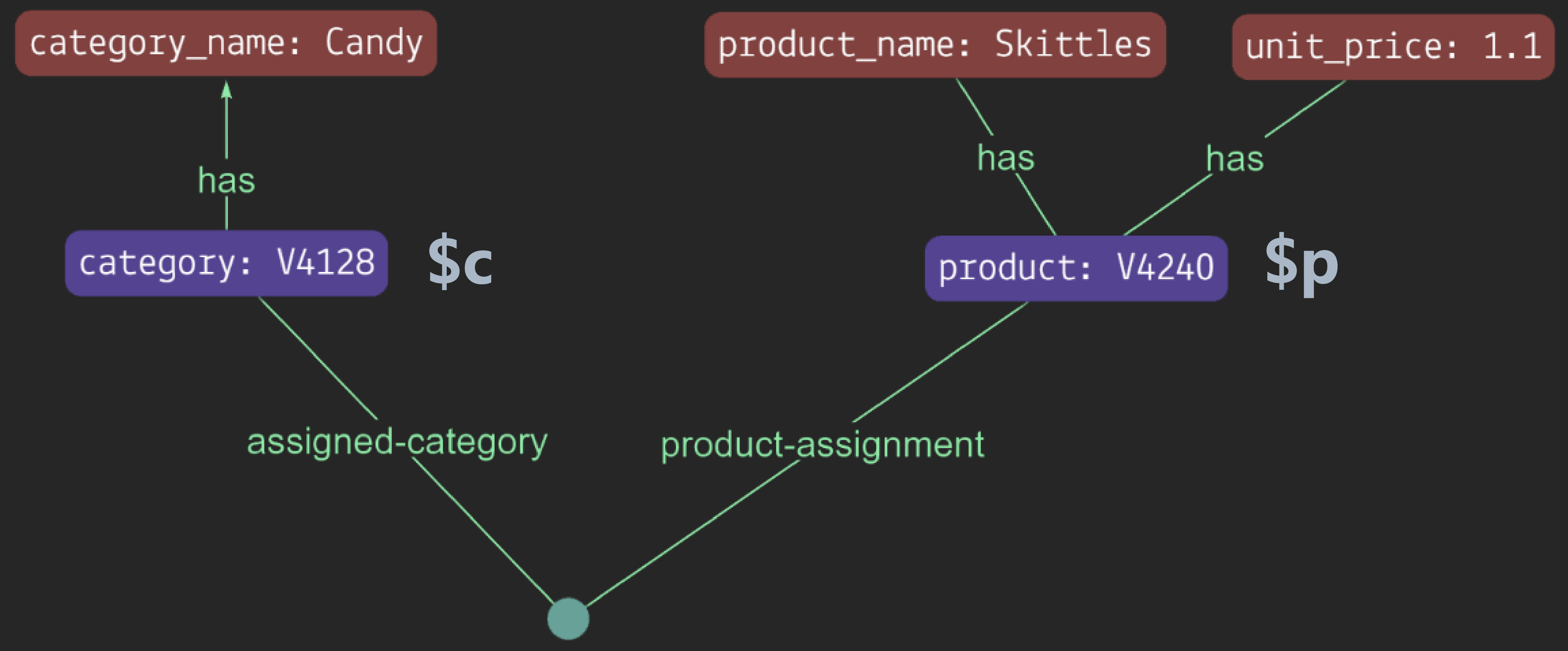
SQL

category_id	category_name
100	Candy

product_id	product_name	category_id	unit_price
99	Skittles	100	1.10

```
INSERT INTO products
   (product_id, product_name, category_id, unit_price)
SELECT 99, 'Skittles', category_id, 1.10
FROM categories
WHERE category_name = 'Candy';
```

Graql



```
match
$c isa category, has category_name "Candy";
insert
$p isa product, has product_name "Skittles", has unit_price 1.10;
(product-assignment: $p, assigned-category: $c)
   isa category-assignment;
```

Writing Data – Associate

SQL

```
INSERT INTO albums
   (album_id, album_name, release_date, artist_id)
SELECT 1, 'Bad', to_date('1987-08-31','YYYY-MM-DD'), artist_id
FROM artists
WHERE artist_name = 'Michael Jackson';
```

Graql

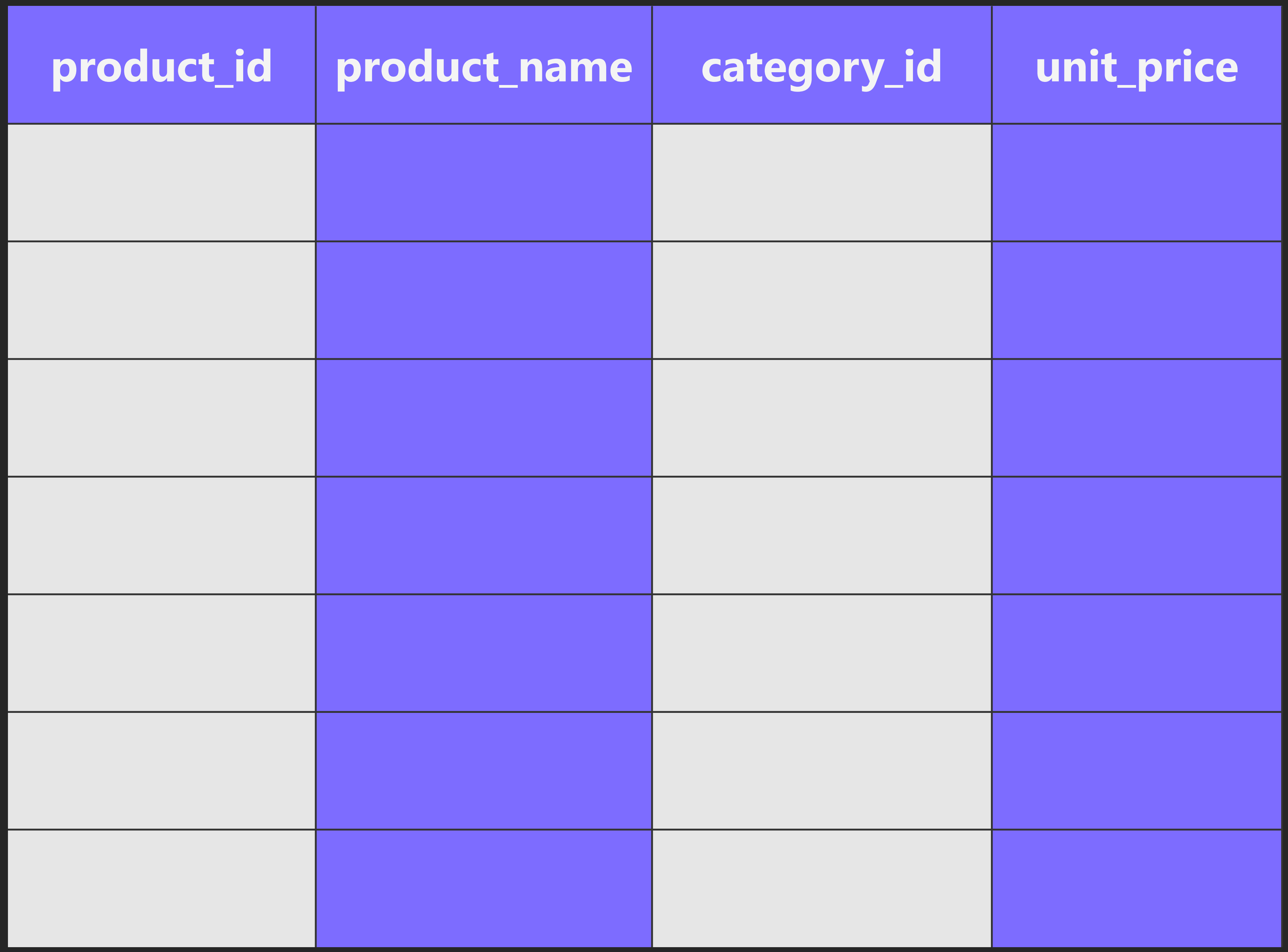
★ Fill in the Blanks ★

```
_____
___ isa artist, ___ artist_name "Michael Jackson";
insert
___ isa album,
   has album_name "Bad",
   has release_date 1987-08-31;
(released_album: $al, releasing_artist: $ar) ___ album-release;
```

★ Fill in the Blanks ★

Writing Data – Associate

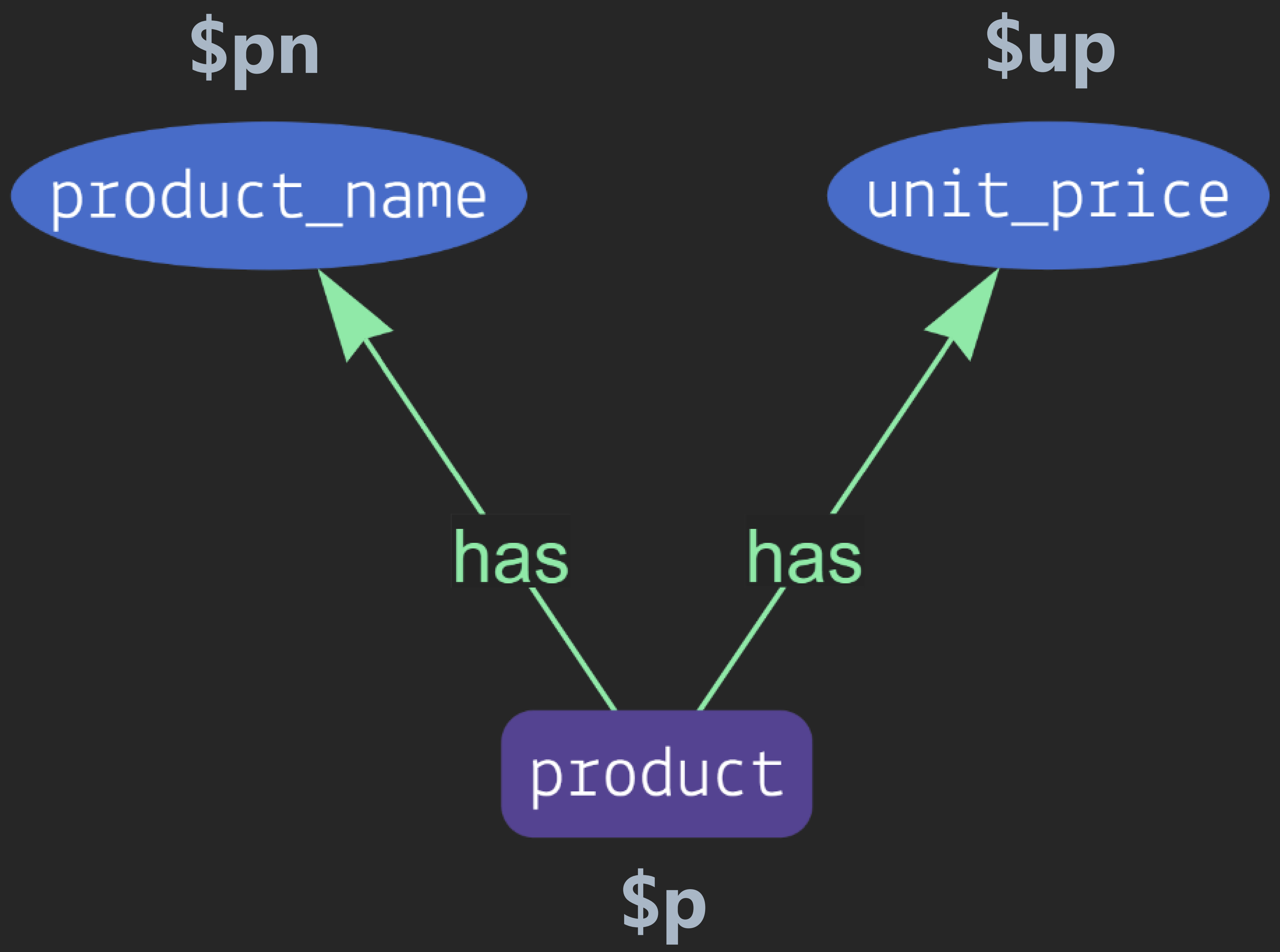
SQL



product_id	product_name	category_id	unit_price

```
SELECT product_name, unit_price
FROM products;
```

Graql



```
match
$p isa product,
   has product_name $pn,
   has unit_price $up;
get $pn, $up;
```

SQL

```
SELECT album_name, release_date
FROM albums;
```

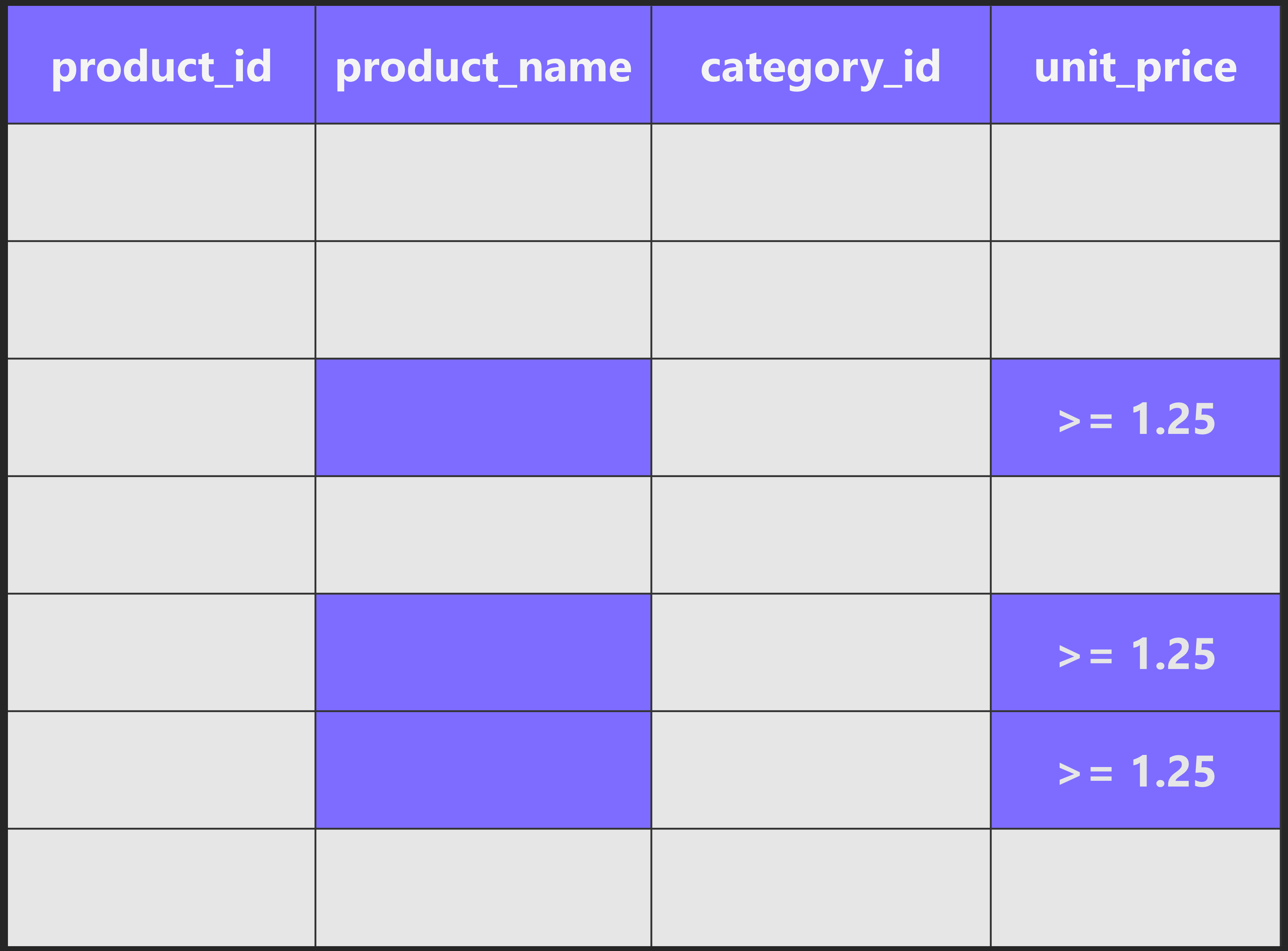
Graql

★ Fill in the Blanks ★

```
match
$a ____ album,
   has album_name $n,
   has release_date $r;
get ___, ___;
```

★ Fill in the Blanks ★

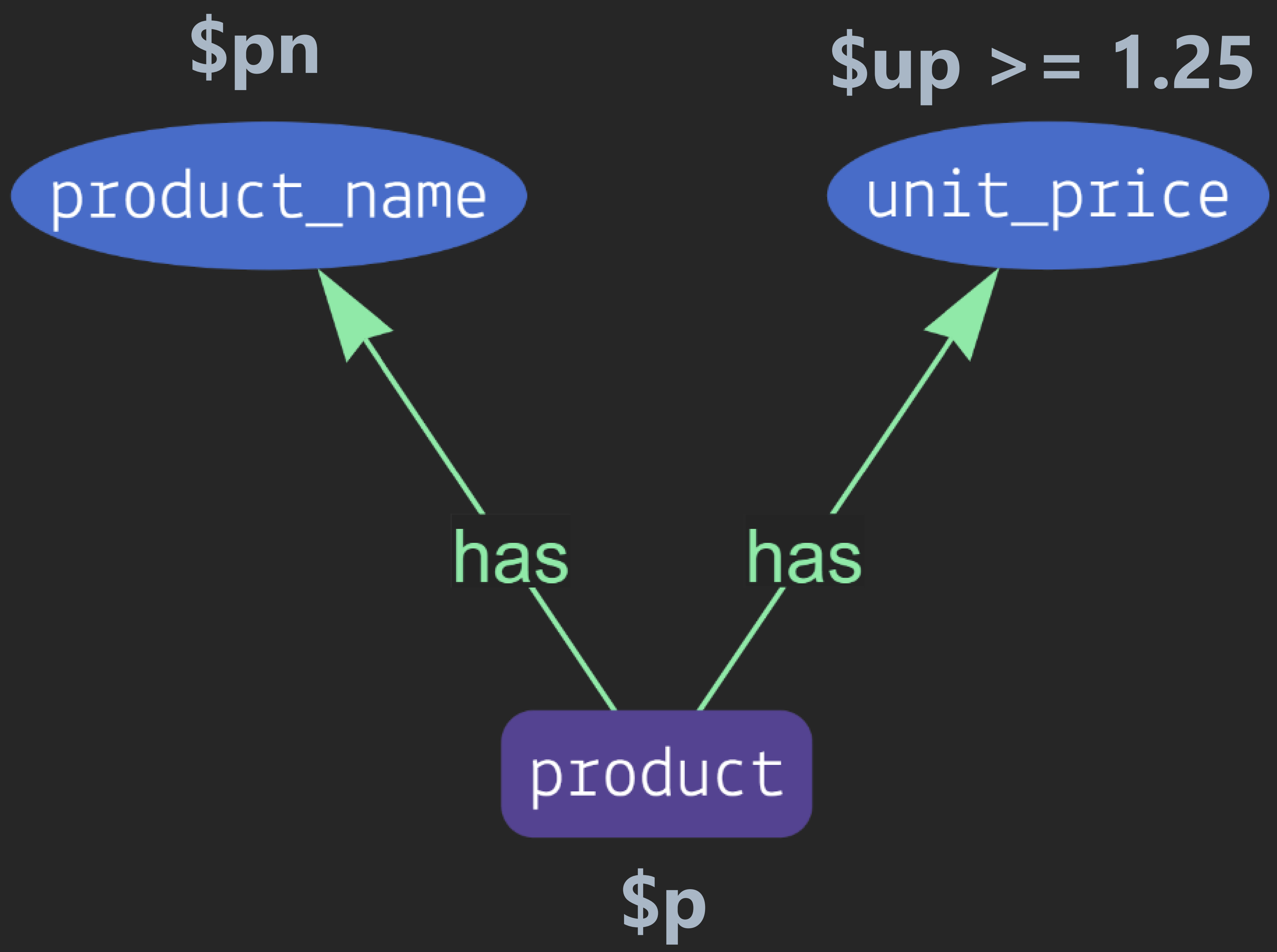
SQL



product_id	product_name	category_id	unit_price
	■		>= 1.25
	■		>= 1.25
	■		>= 1.25

```
SELECT product_name, unit_price
FROM products
WHERE unit_price >= 1.25;
```

Graql



```
match
$p isa product,
   has product_name $pn,
   has unit_price $up;
$up >= 1.25;
get $pn, $up;
```

Reading Data – Restrict

SQL

```
SELECT album_name, release_date
FROM albums
WHERE release_date < to_date('1990-01-01','YYYY-MM-DD');
```

Graql

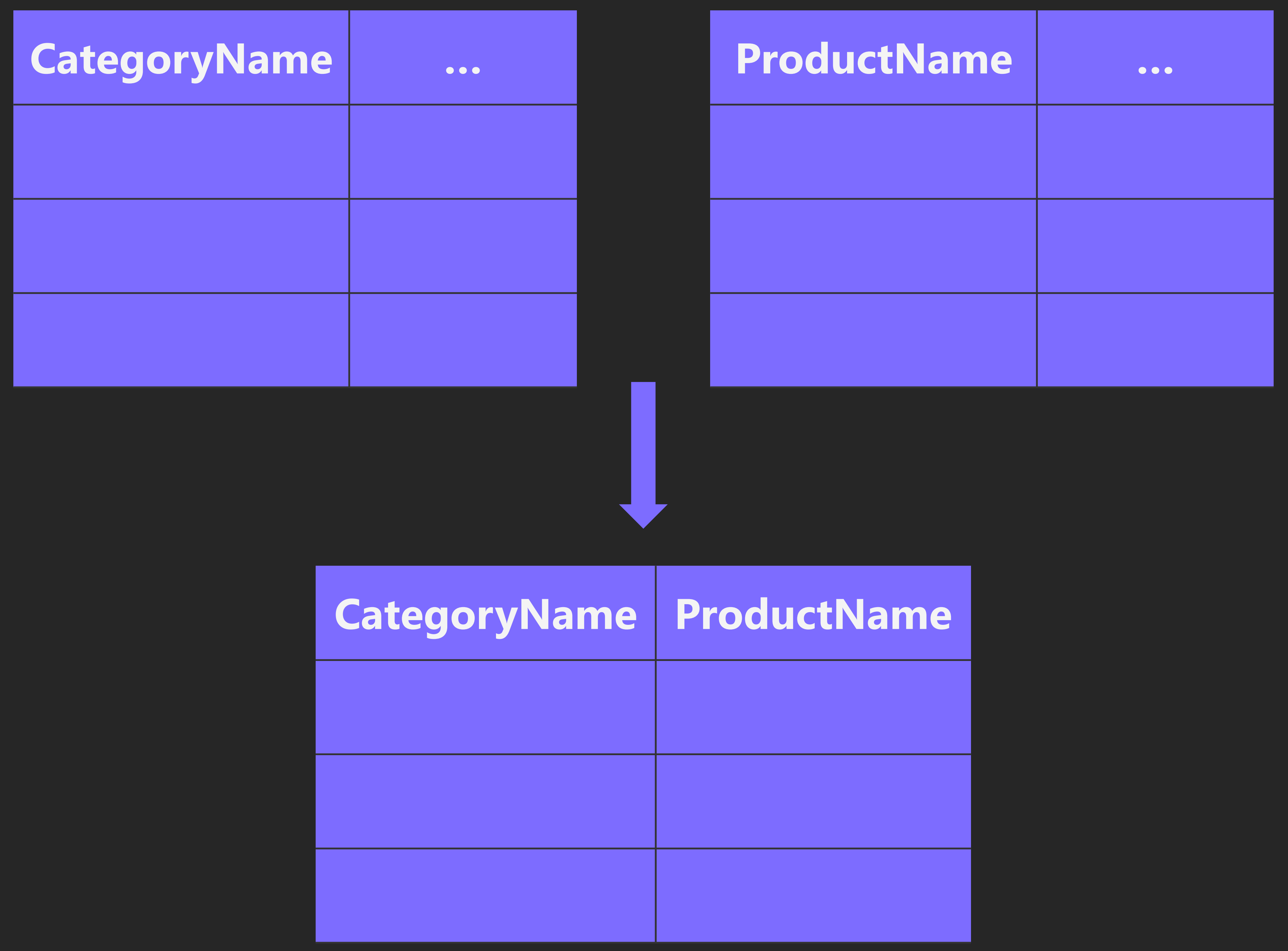
★ Fill in the Blanks ★

```
match
$a isa album,
   has album_name $n,
   has release_date $r;
$r < 1990-01-01;
____ ___, ___;
```

★ Fill in the Blanks ★

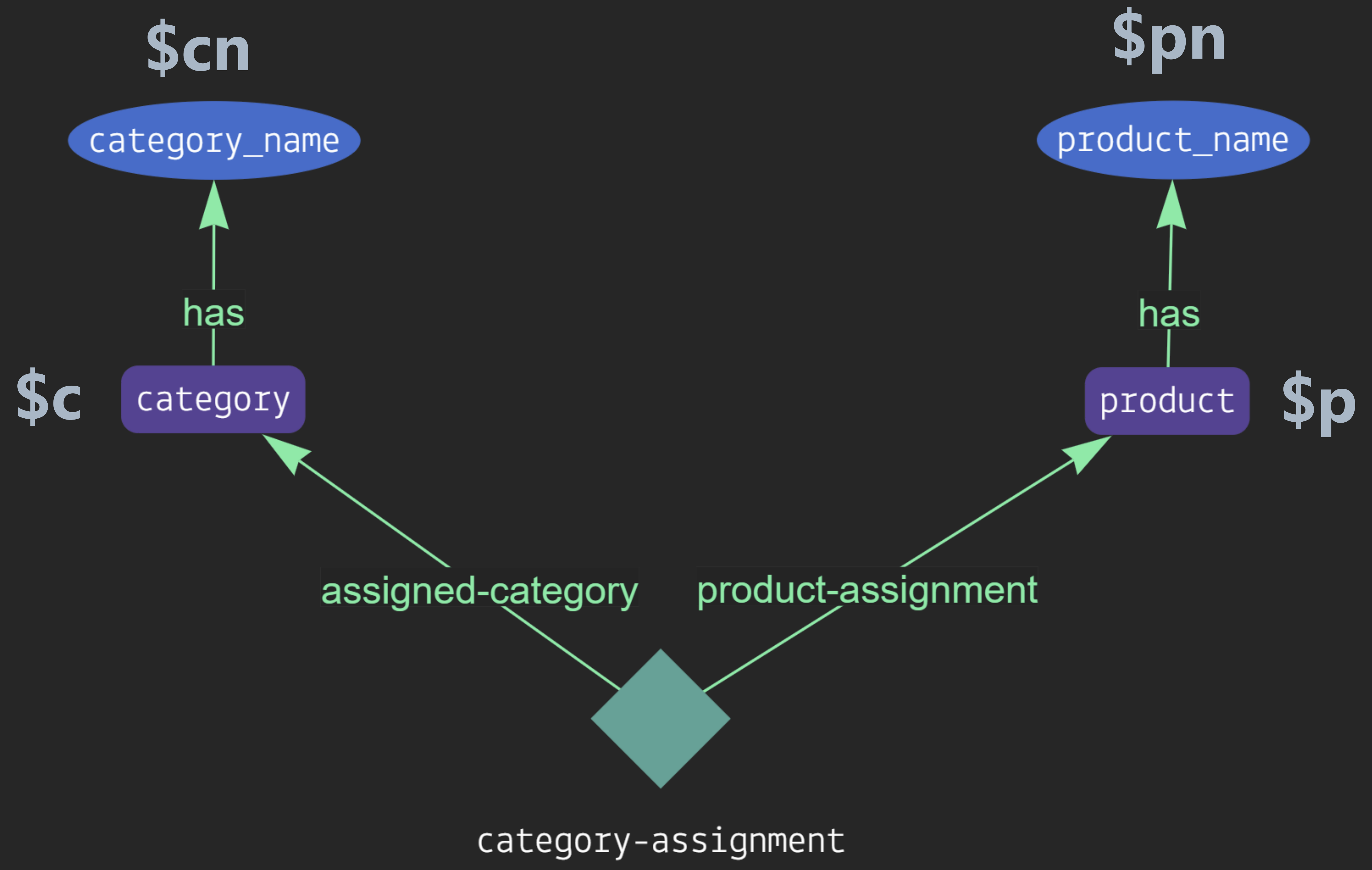
Reading Data – Restrict

SQL



```
SELECT category_name, product_name
FROM categories
INNER JOIN products
   ON categories.category_id = products.category_id;
```

Graql



```
match
$c isa category, has category_name $cn;
$p isa product, has product_name $pn;
($c, $p) isa category-assignment;
get $cn, $pn;
```

Reading Data – Inner Join

SQL

```
SELECT artist_name, album_name
FROM artists
INNER JOIN albums
   ON artists.artist_id = albums.artist_id;
```

Graql

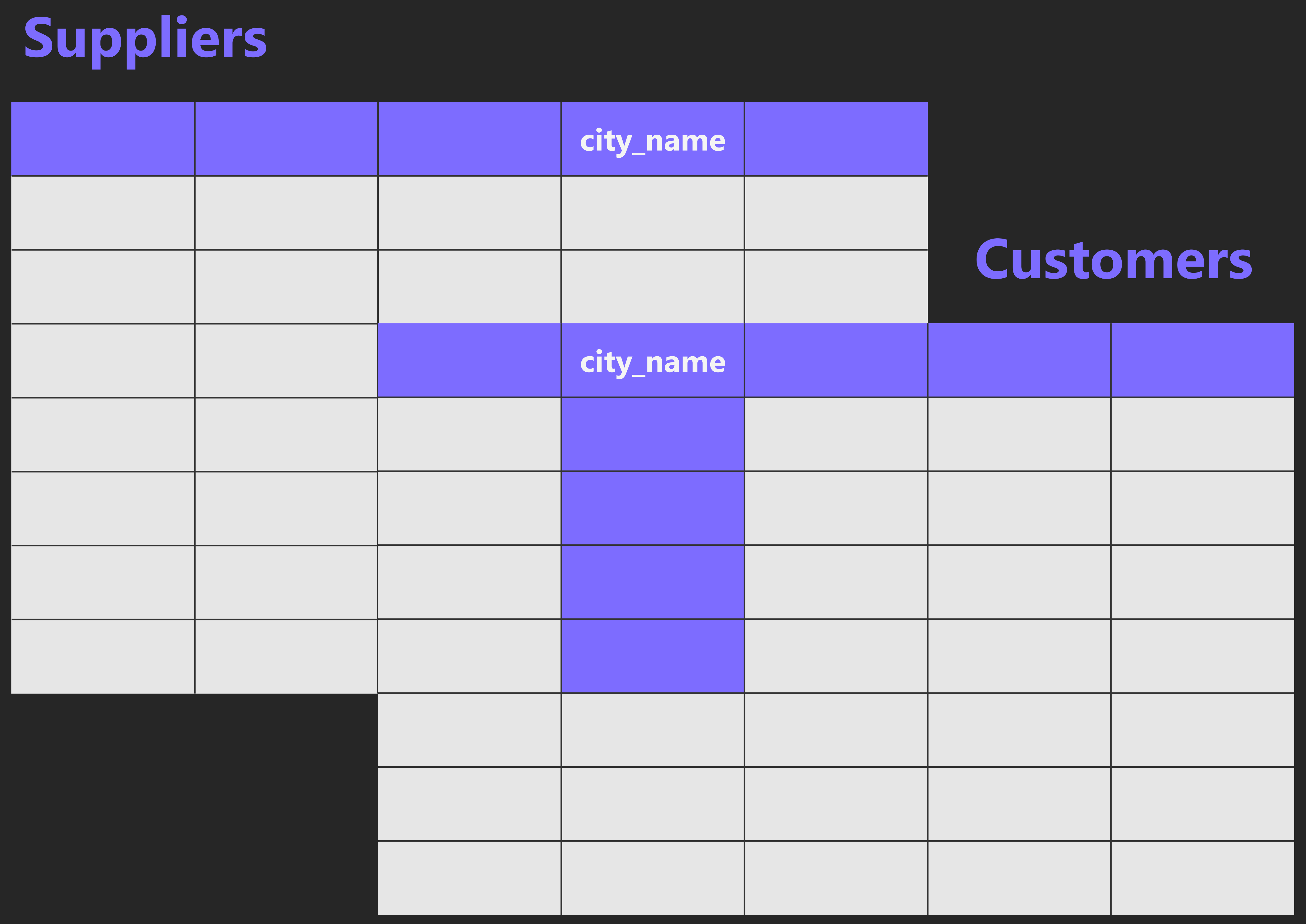
★ Fill in the Blanks ★

```
match
$ar isa artist,
   has artist_name $ar_name;
$al isa album,
   has album_name $al_name;
(___, ___) isa album-release;
get $ar_name, $al_name;
```

★ Fill in the Blanks ★

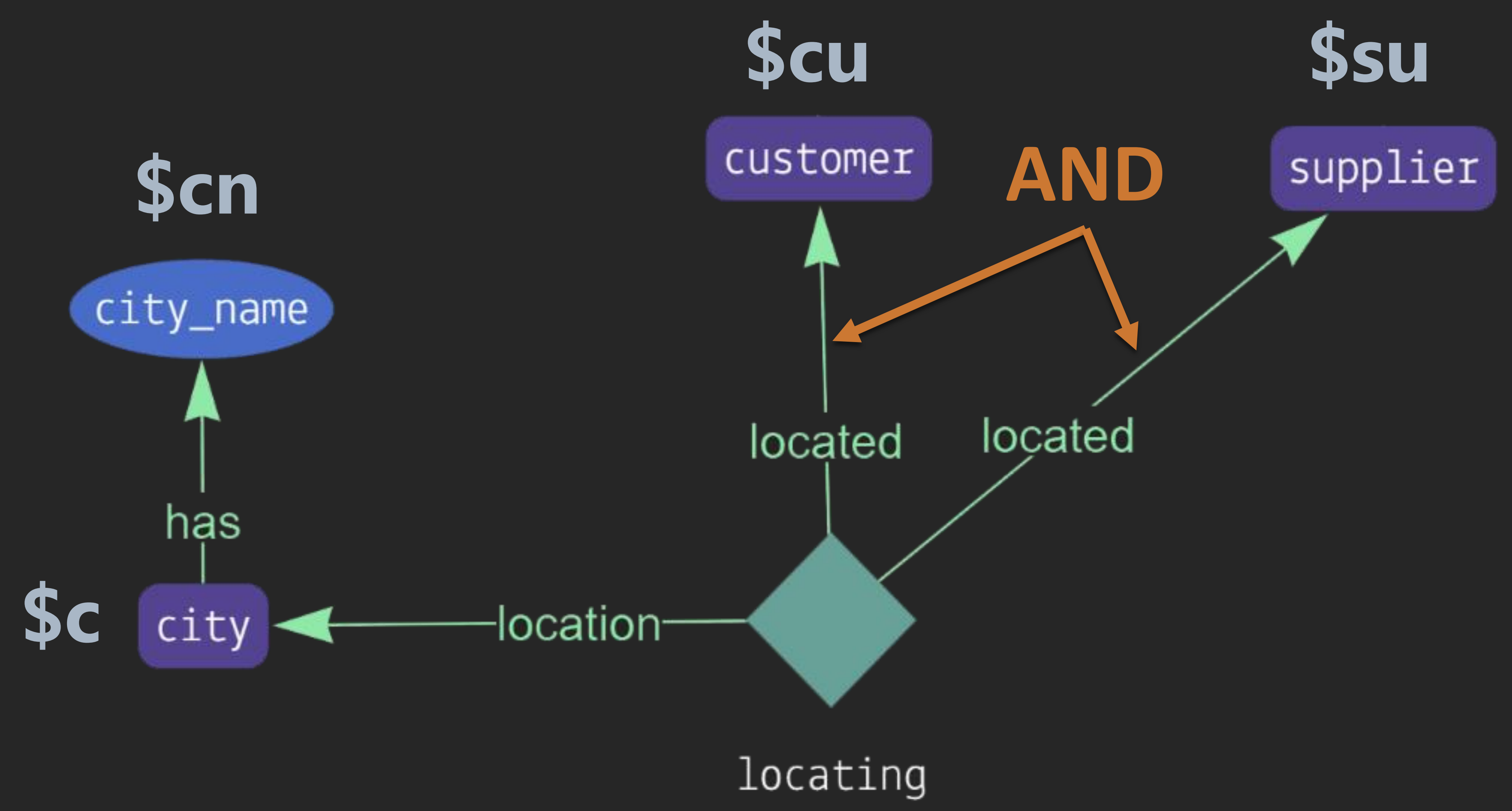
Reading Data – Inner Join

SQL



```
SELECT city_name
FROM customers
INTERSECT
SELECT city_name
FROM suppliers;
```

Graql



```
match
$c isa city, has city_name $cn;
$cu isa customer; (location: $c, $cu);
$su isa supplier; (location: $c, $su);
get $cn;
```

Reading Data – Intersect

SQL

```
SELECT album_name
FROM albums
INNER JOIN artists on albums.artist_id = artists.artist_id
WHERE artist_name = 'Michael Jackson’
INTERSECT
SELECT album_name
FROM albums
WHERE release_date >= to_date('2000-01-01','YYYY-MM-DD');
```

Graql

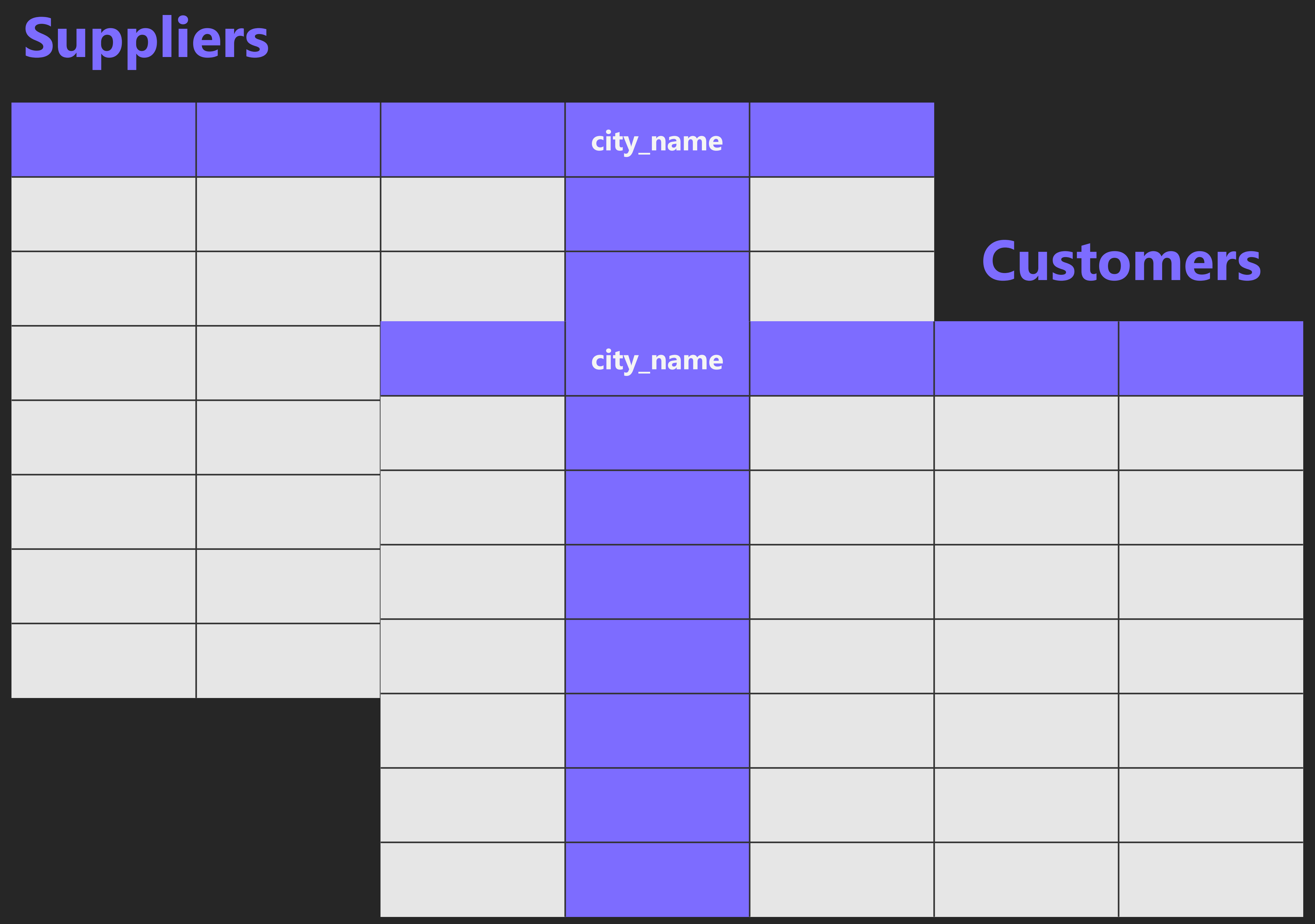
★ Fill in the Blanks ★

```
______
$al ___ album, has album_name __;
$ar isa artist, ___ artist_name "Michael Jackson";
(released_album: ___, releasing_artist: $ar) isa album-release;
$al has release_date >= __________;
get $n;
```

★ Fill in the Blanks ★

Reading Data – Intersect

SQL



Suppliers

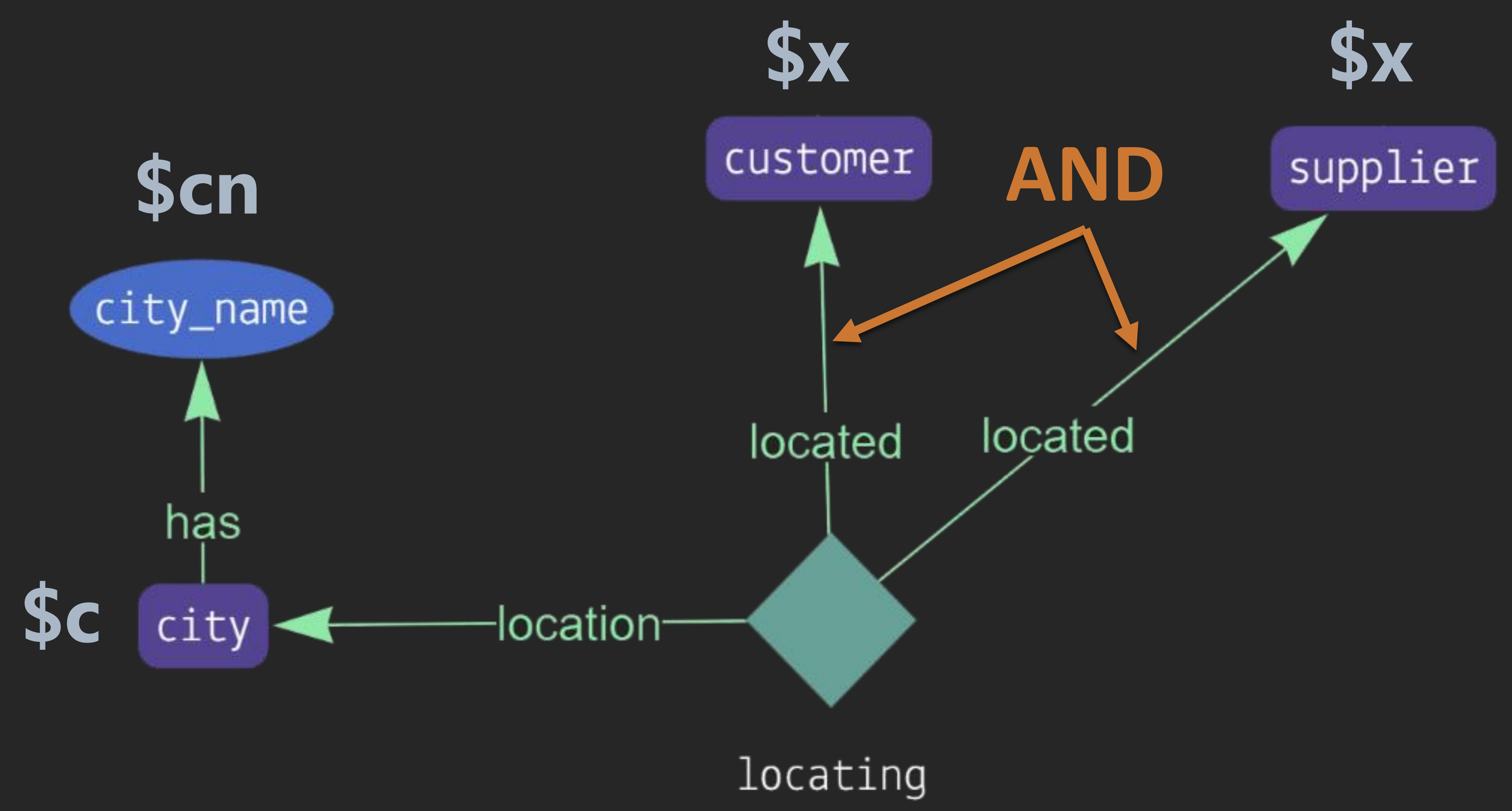
city_name

Customers

city_name

```
SELECT city_name
FROM customers
UNION
SELECT city_name
FROM suppliers;
```

Graql



$cn

city_name

has

$c

city

location

locating

located

located

$x

customer

AND

$x

supplier

```
match
$c isa city, has city_name $cn;
{$x isa supplier;} or {$x isa customer;};
(location: $c, $x);
get $cn;
```

Reading Data – Union

SQL

```
SELECT album_name
FROM albums
INNER JOIN artists on albums.artist_id = artists.artist_id
WHERE artist_name = 'Michael Jackson’
UNION
SELECT album_name
FROM albums
WHERE release_date >= to_date('2000-01-01','YYYY-MM-DD');
```

Graql

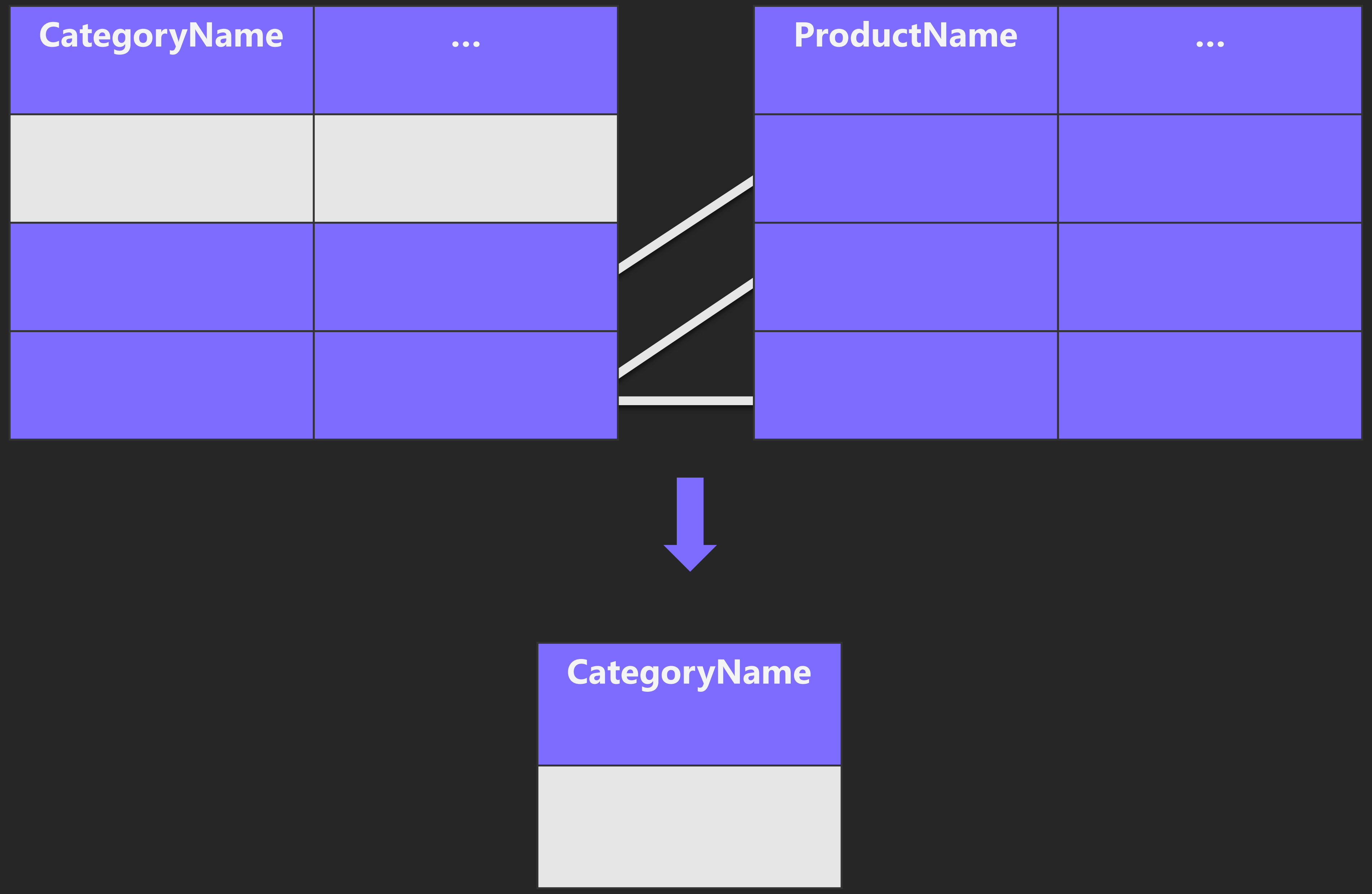
★ Fill in the Blanks ★

```
match
____ ____ ______, ____ __________ $n;
{
   $ar isa artist, has artist_name "Michael Jackson";
   (released_album: $al, releasing_artist: $ar) isa album-release;
} __ {
   $al has release_date >= 2000-01-01;
};
get $n;
```

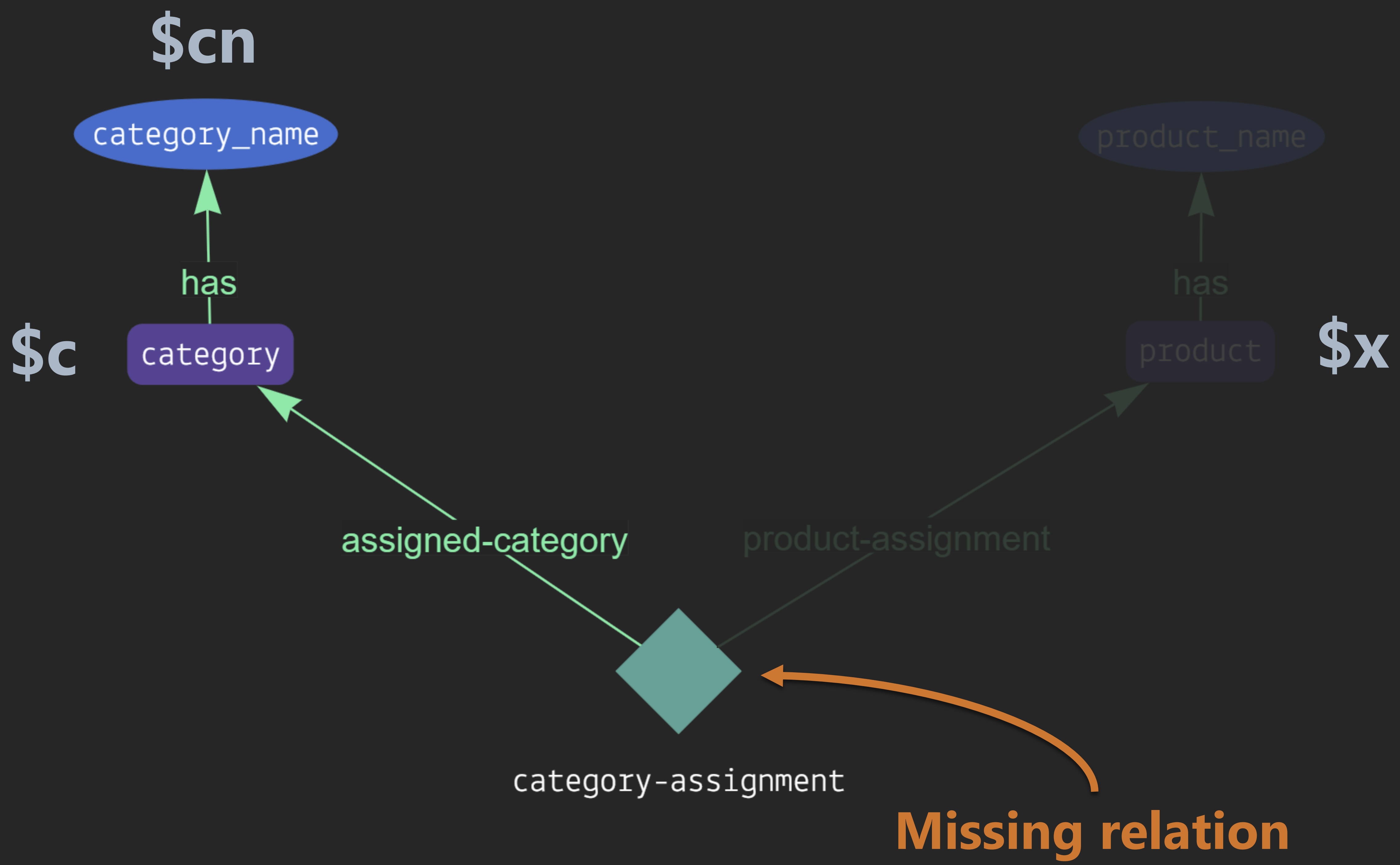
★ Fill in the Blanks ★

Reading Data – Union

SQL



Graql



```sql
SELECT category_name
FROM categories
LEFT JOIN products
   ON categories.category_id = products.category_id
WHERE products.category_id IS NULL;
```

```
match
$c isa category,
   has category_name $cn;
not { ($c, $x) isa category-assignment; };
get $cn;
```

Reading Data – Left Join

SQL

```sql
SELECT artist_name
FROM artists
LEFT JOIN albums
   ON artists.artist_id = albums.artist_id
WHERE albums.artist_id IS NULL;
```

Graql

★ Fill in the Blanks ★

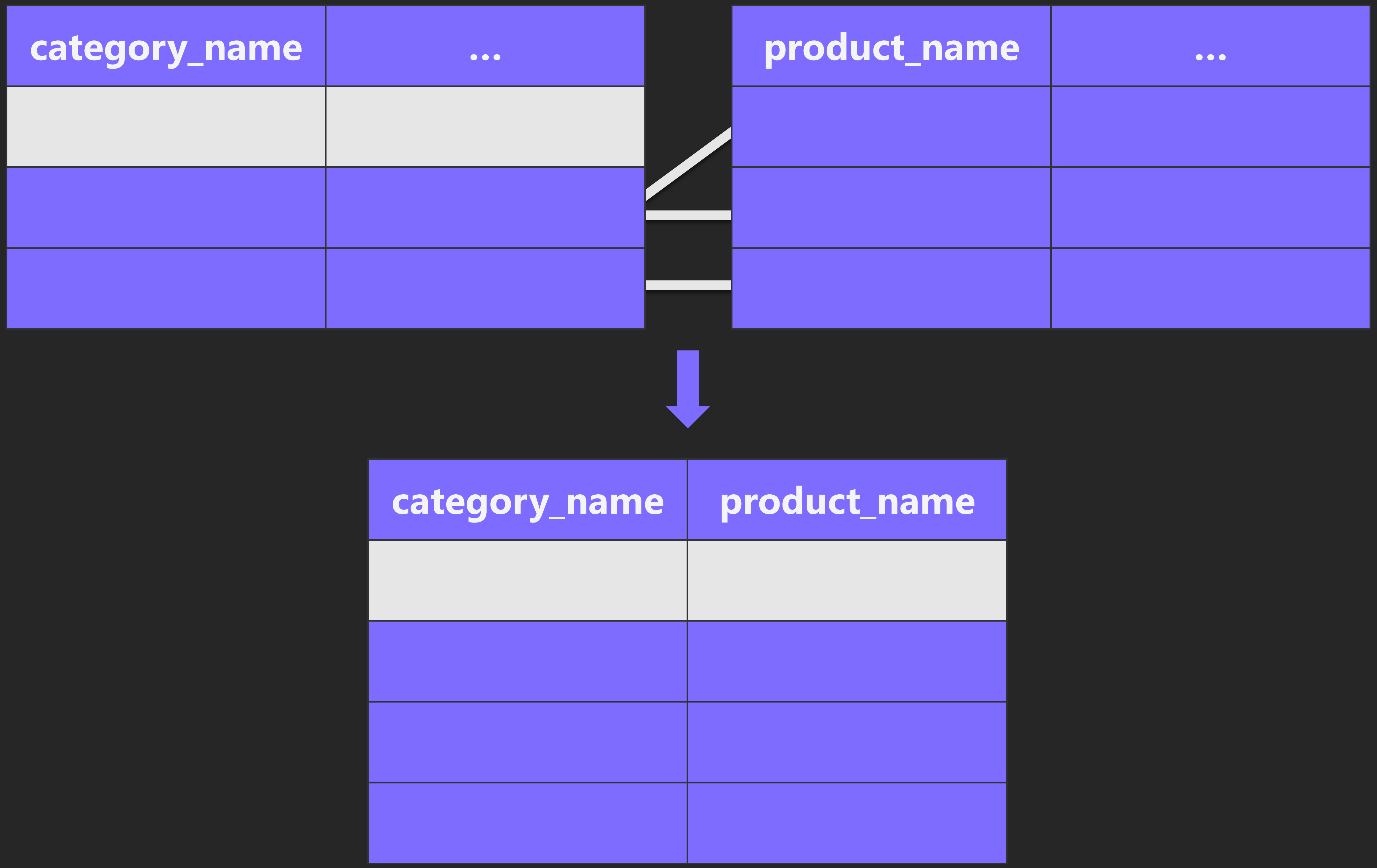
```
______
$ar isa artist,
   has artist_name $ar_name;
____ {
   ($ar, $x) ____ album-release;
};
____ $ar_name;
```

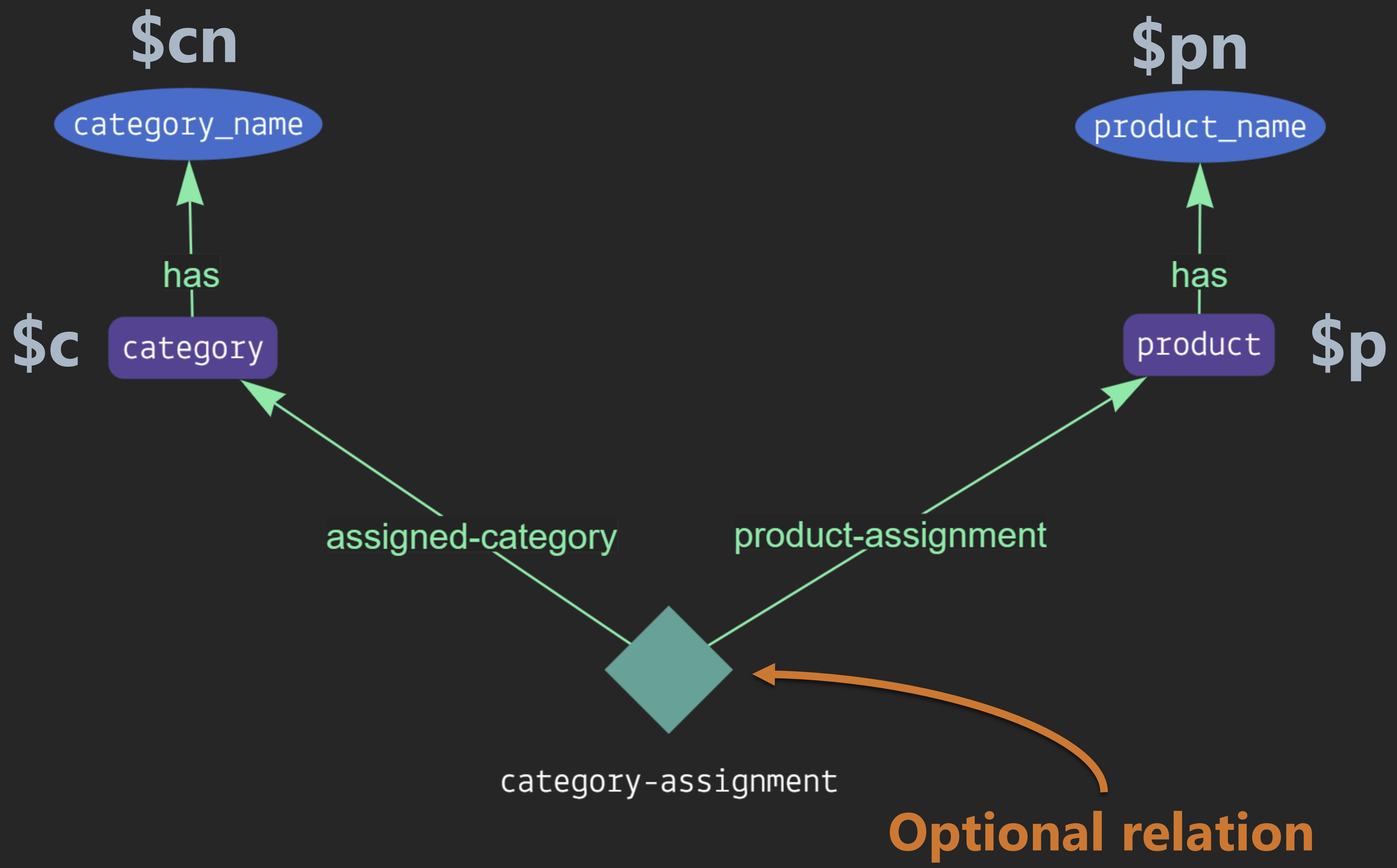
★ Fill in the Blanks ★

Reading Data – Left Join

SQL

Graql





```
SELECT category_name
FROM categories
LEFT JOIN products
   ON categories.category_id = products.category_id
WHERE products.category_id IS NULL;
```

```
match
$c isa category, has category_name $cn;
$p isa product, has product_name $pn;
{  ($c, $p) isa category-assignment; }
or { not { ($c, $p) isa category-assignment; }; };
get $cn, $pn;
```

Reading Data – Full Outer Join

SQL

```
SELECT artist_name, album_name
FROM artists
FULL OUTER JOIN albums
   ON artists.artist_id = albums.artist_id;
```

Graql

★ Fill in the Blanks ★

```
match
$ar isa artist, has artist_name $ar_name;
$al ____ album, has album_name $al_name;
{
  ($ar, $al) isa album-release;
} __ {
   ____ {
     ($ar, $al) isa _______________;
   };
};
get $ar_name, _________;
```

★ Fill in the Blanks ★

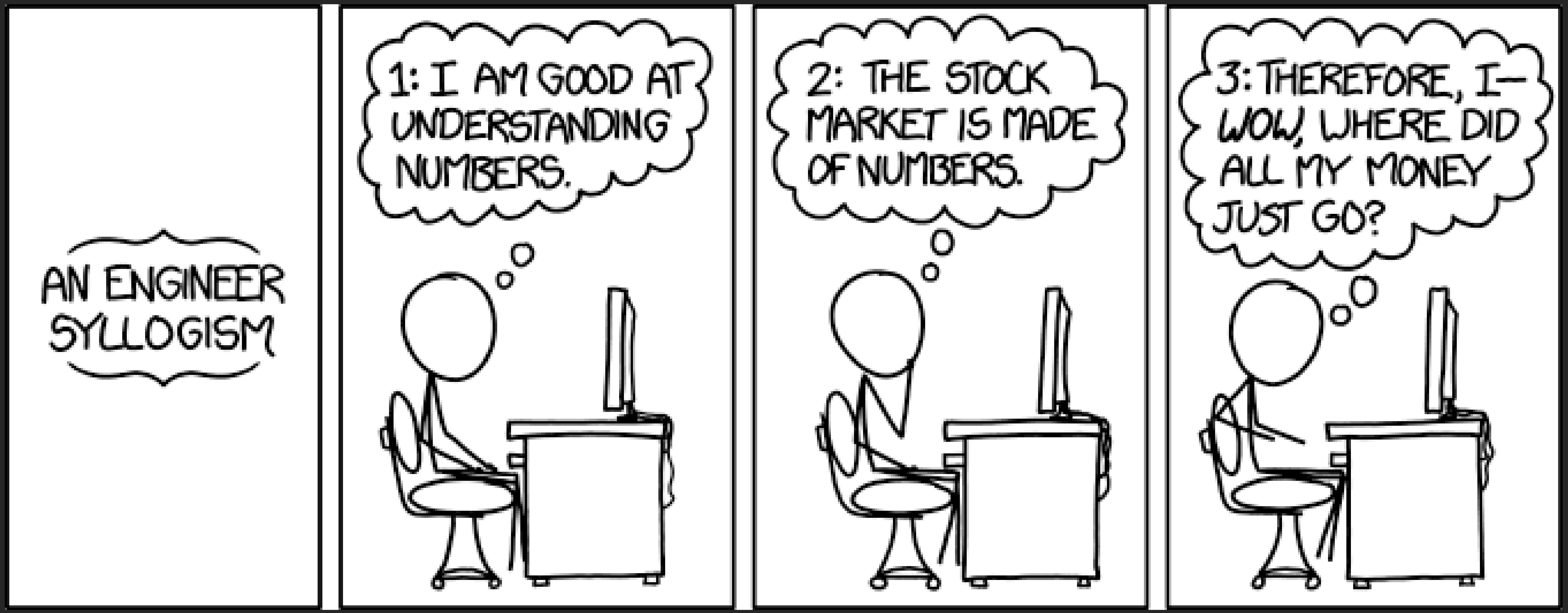
Reading Data – Full Outer Join

Reading Data – Inference

- All men are mortal
- Socrates is a man
- Therefore, Socrates is mortal

```
men-are-mortal sub rule,
when {
   $man isa man;
},
then {
   $man isa mortal;
};

insert $m isa man, has name "Socrates";
match $mortal isa mortal; get;
```

Reading Data – Inference


AN ENGINEER SYLLOGISM
1: I AM GOOD AT UNDERSTANDING NUMBERS.
2: THE STOCK MARKET IS MADE OF NUMBERS.
3: THEREFORE, I— WOW, WHERE DID ALL MY MONEY JUST GO?

https://xkcd.com/1570/

- All women are mortal
- Socrates is a man
- Therefore, Socrates is mortal

Reading Data – Inference

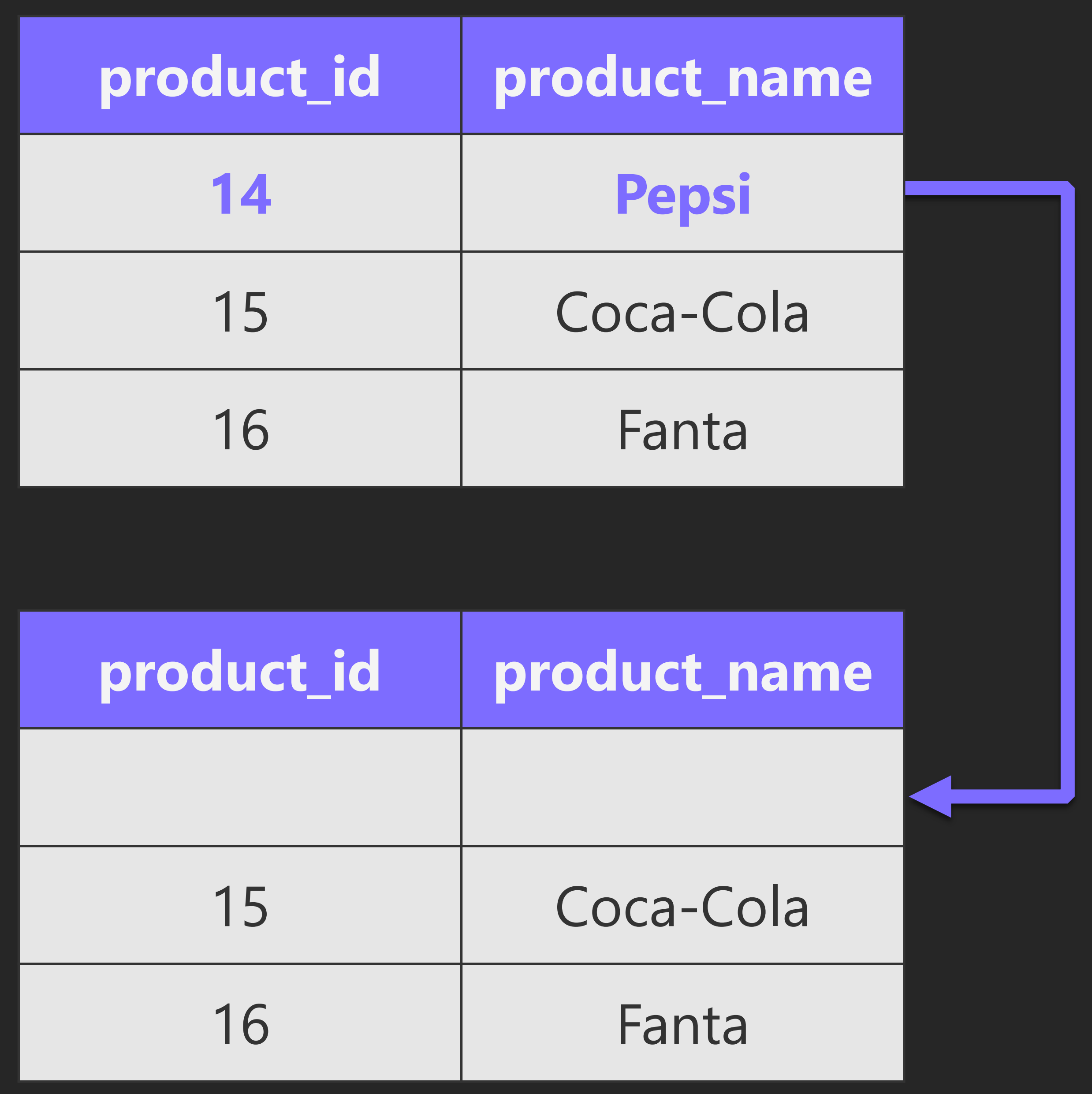
- When
  - Texas *($b)* is located in USA *($a)*
  - Dallas *($c)* is located in Texas *($b)*

- Then
  - Dallas *($c)* is located in USA *($a)*

```
transitive-location sub rule,
when {
   (location: $a, located: $b) isa locating;
   (location: $b, located: $c) isa locating;
}, then {
   (location: $a, located: $c) isa locating;
};
```

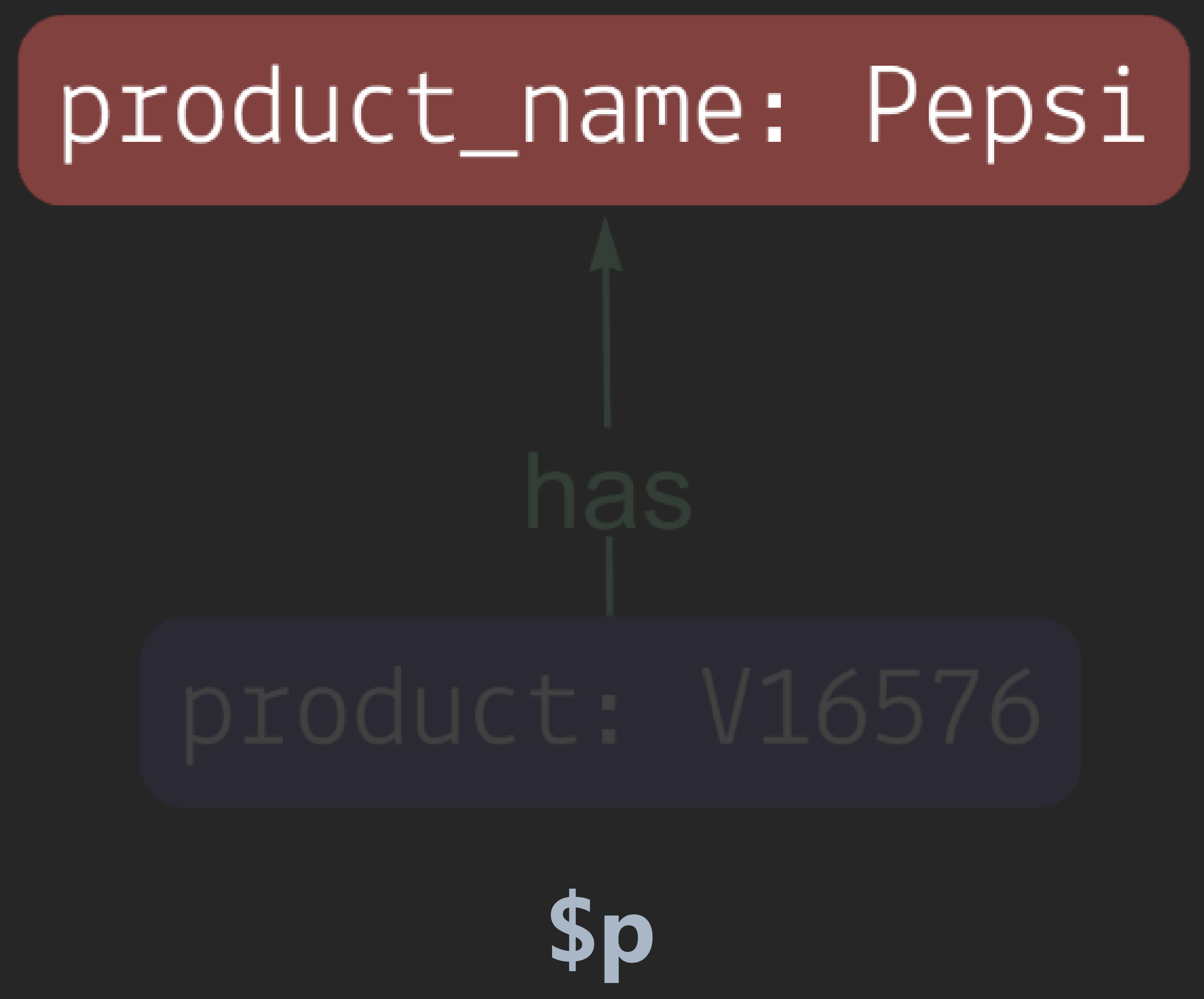
SQL

Graql



product_id	product_name
14	Pepsi
15	Coca-Cola
16	Fanta

product_id	product_name
15	Coca-Cola
16	Fanta



product_name: Pepsi

has

product: V16576

$p

```
DELETE FROM products
WHERE product_name = 'Pepsi';
```

```
match
$p isa product, has product_name "Pepsi";
delete $p;
```

Deleting Data

SQL

```
DELETE
FROM albums
USING artists
WHERE albums.artist_id = artists.artist_id
AND artists.artist_name = 'Michael Jackson’
AND albums.album_name = 'Bad';
```

Graql

★ Fill in the Blanks ★

```
______
$ar isa artist, has artist_name "Michael Jackson";
$al isa album, has album_name "Bad";
($ar, $al) isa album-release;
_______ ___;
```

★ Fill in the Blanks ★

Deleting Data

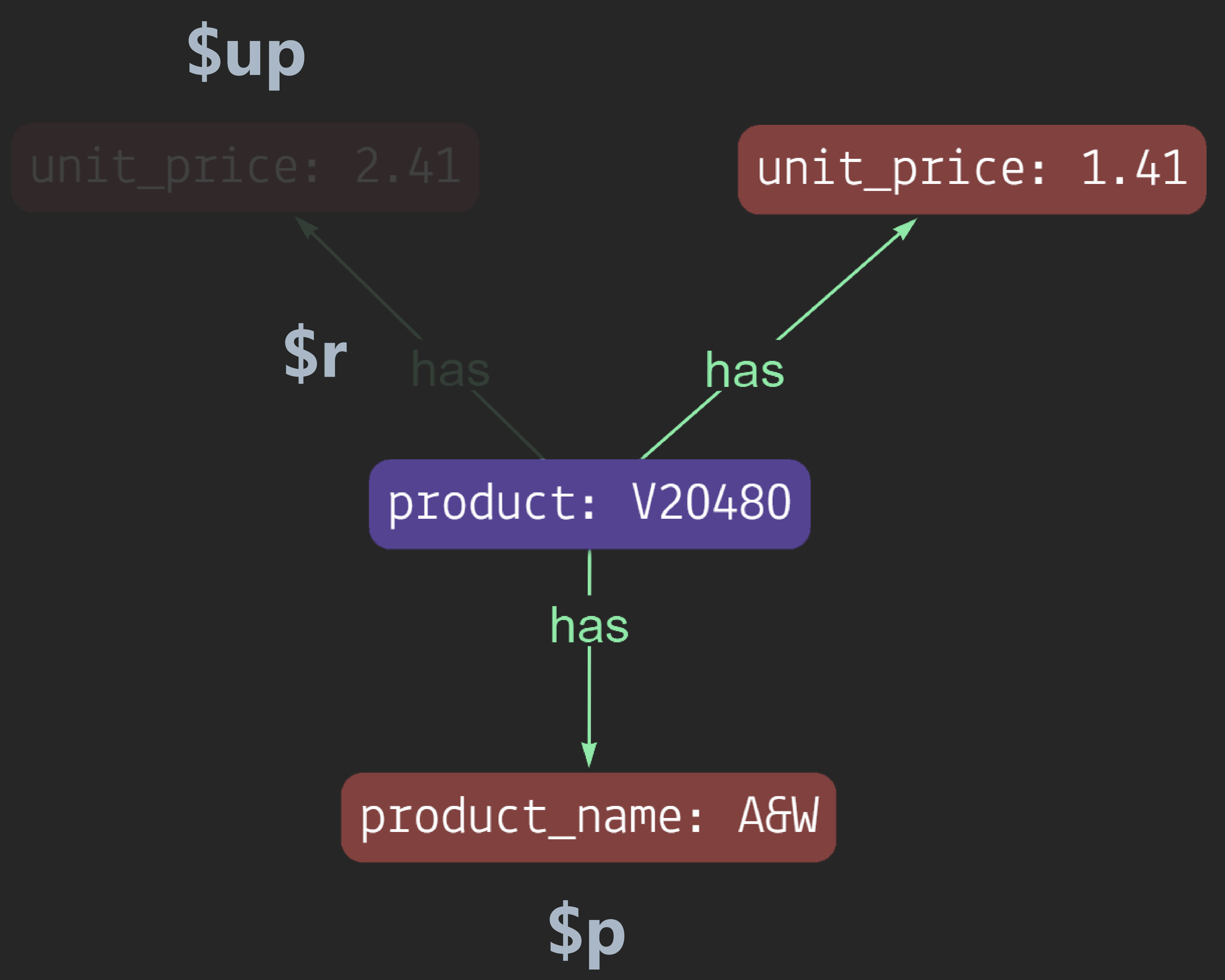
SQL

product_id	product_name	unit_price
17	Sprite	$1.55
18	7 UP	$1.68
19	A&W	**$2.41**

product_id	product_name	unit_price
17	Sprite	$1.55
18	7 UP	$1.68
19	A&W	**$1.41**

```
UPDATE products
SET unit_price = 1.41
WHERE product_name = 'A&W';
```

Graql



```
match $p isa product,
   has product_name "A&W", has unit_price $up via $r;
delete $r;

match $p isa product, has product_name "A&W";
insert $p has unit_price 1.41;
```

Modifying Data

SQL

```
UPDATE artists
SET artist_name = 'Michael Joseph Jackson’
WHERE artist_name = 'Michael Jackson';
```

Graql

★ Fill in the Blanks ★

```
______ $ar isa artist, has artist_name "Michael Jackson";
_______ $ar has artist_name "Michael Joseph Jackson";

______ $ar isa artist, has artist_name "Michael Jackson" via $r;
_______ $r;
```

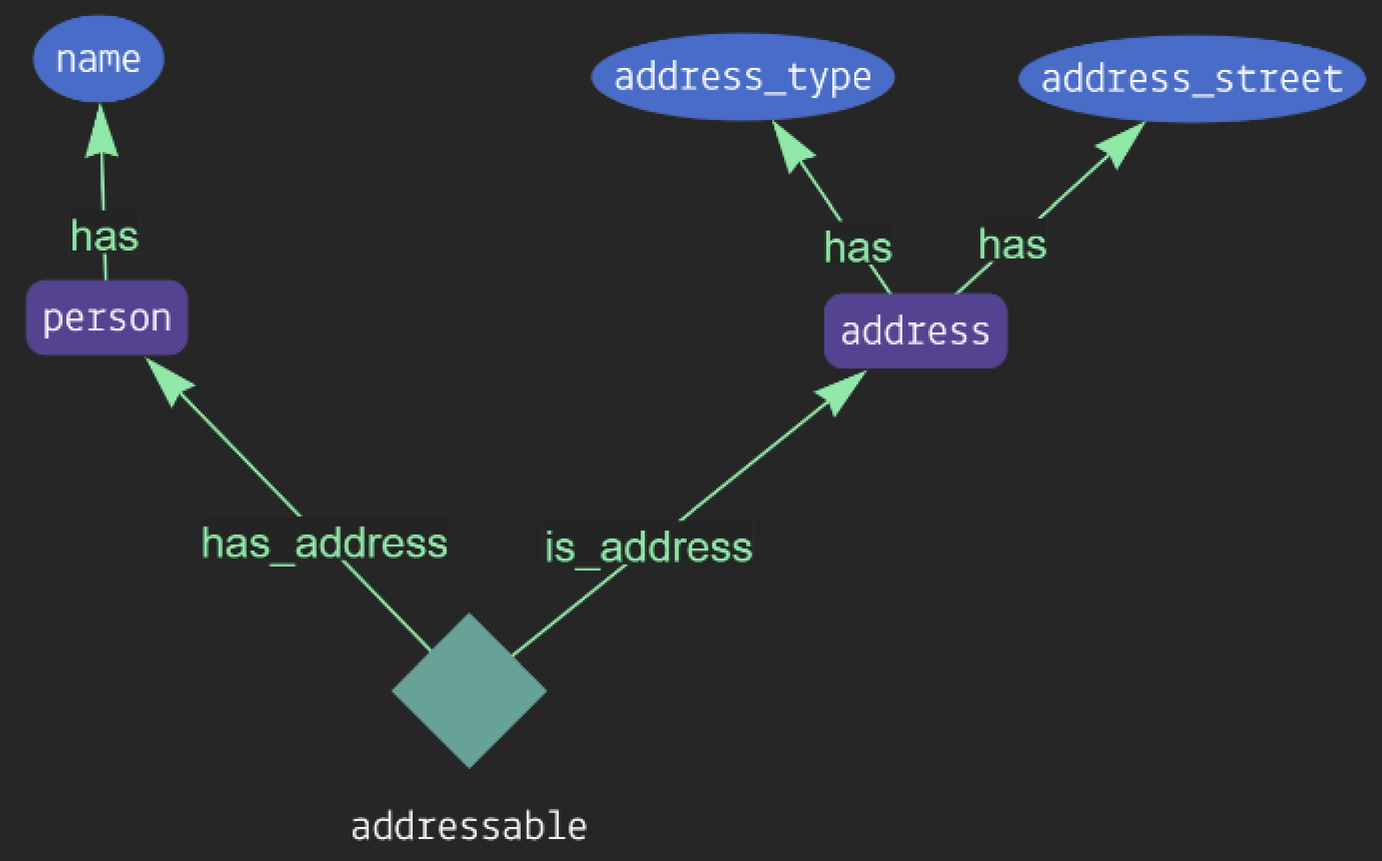
★ Fill in the Blanks ★

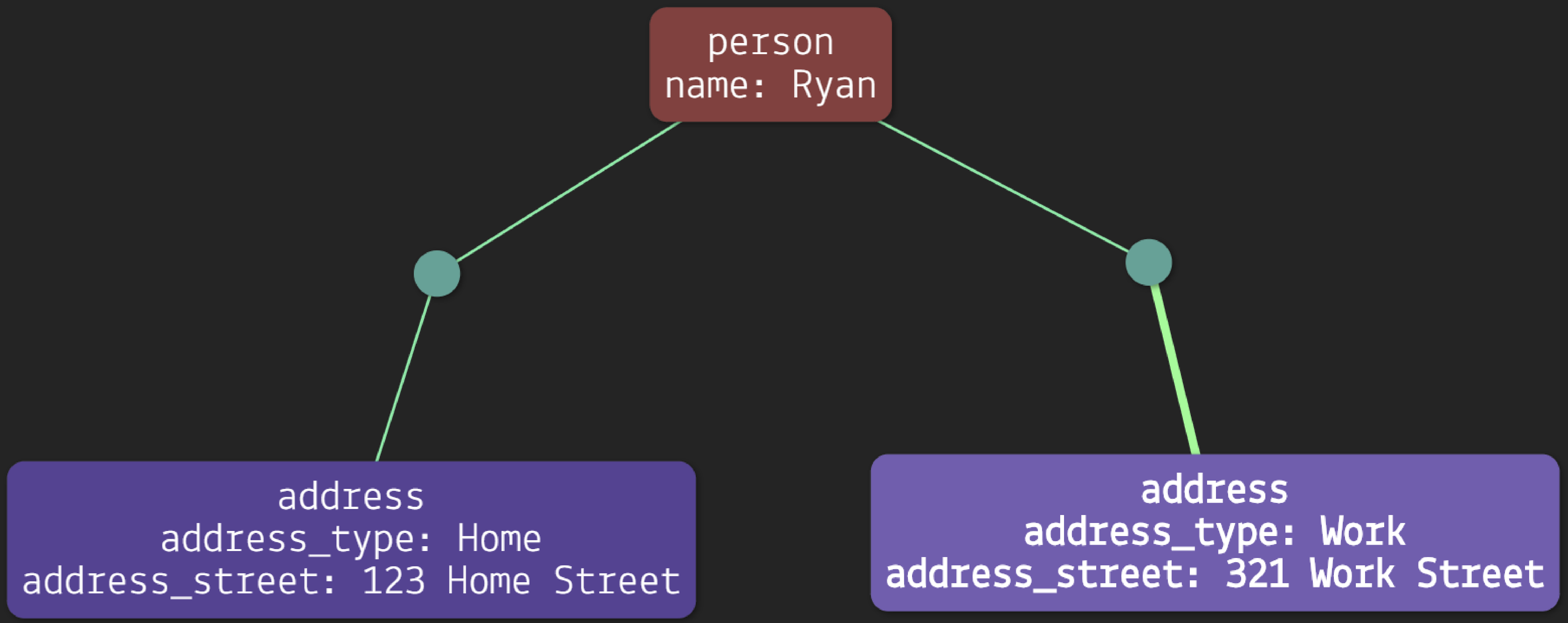
Modifying Data

Exercises

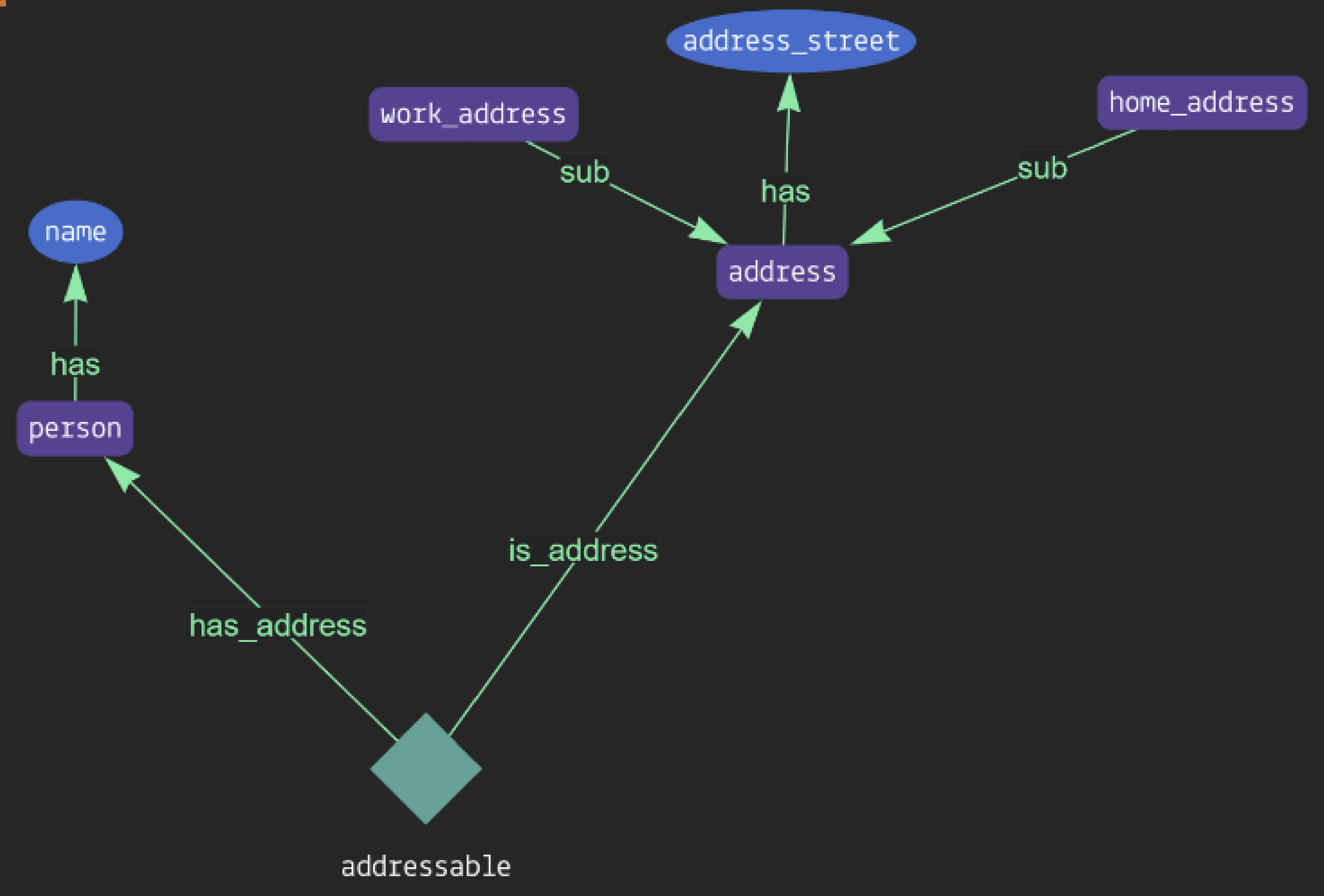
Exercise – Which is Better?

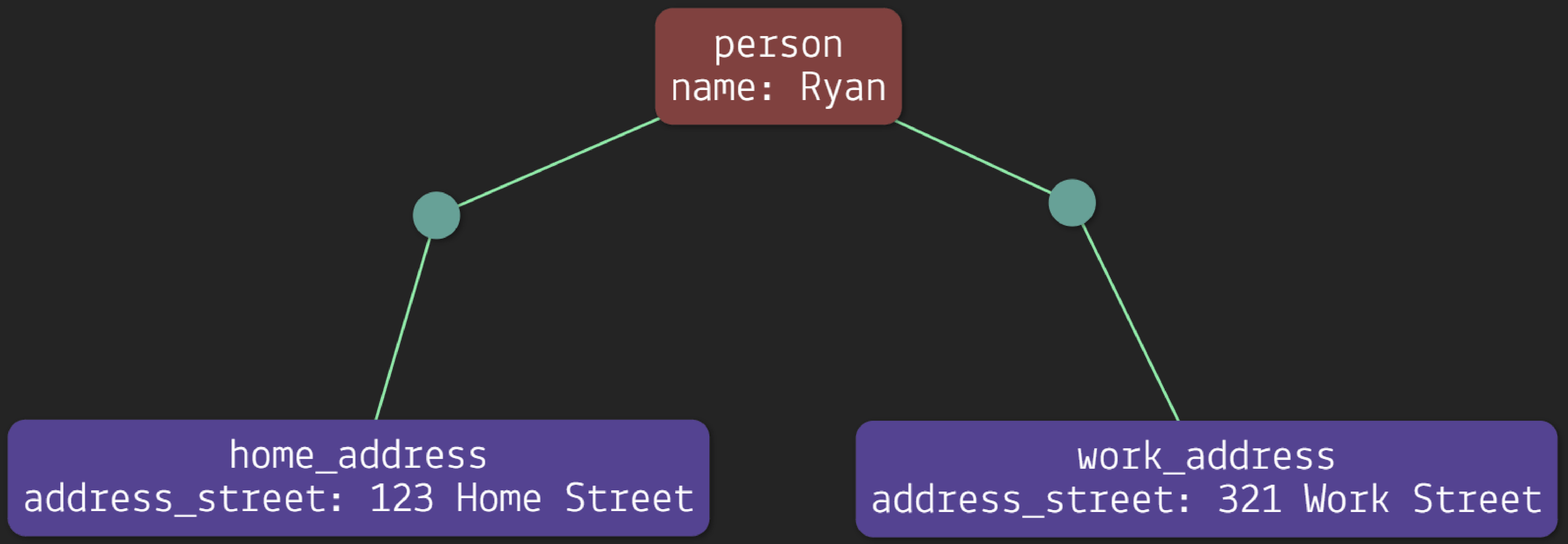
Option A





Option B





person
name: Ryan
home_address
address_street: 123 Home Street
work_address
address_street: 321 Work Street

Exercise – Will This Work?

```
################################################## SCHEMA ##################################################
define
product sub entity,
   key product_name;
city sub entity,
   has city_name,
   plays location,
   plays located;
locating sub relation,
   relates location,
   relates located;
product_name sub attribute, datatype string;
city_name sub attribute, datatype string;

################################################### DATA ###################################################
insert
$bacon isa product, has product_name "Bacon";
$toronto isa city, has city_name "Toronto";
(location: $bacon, located: $toronto) isa locating;
```

Exercise – Convert Tables

Objectives

1. Convert the following tables to an equivalent Grakn schema
2. Populate the knowledge base with the equivalent data
3. Write a query to get the primary language of Argentina

Countries

country_id	country_name	population
1	Argentina	45,000,000

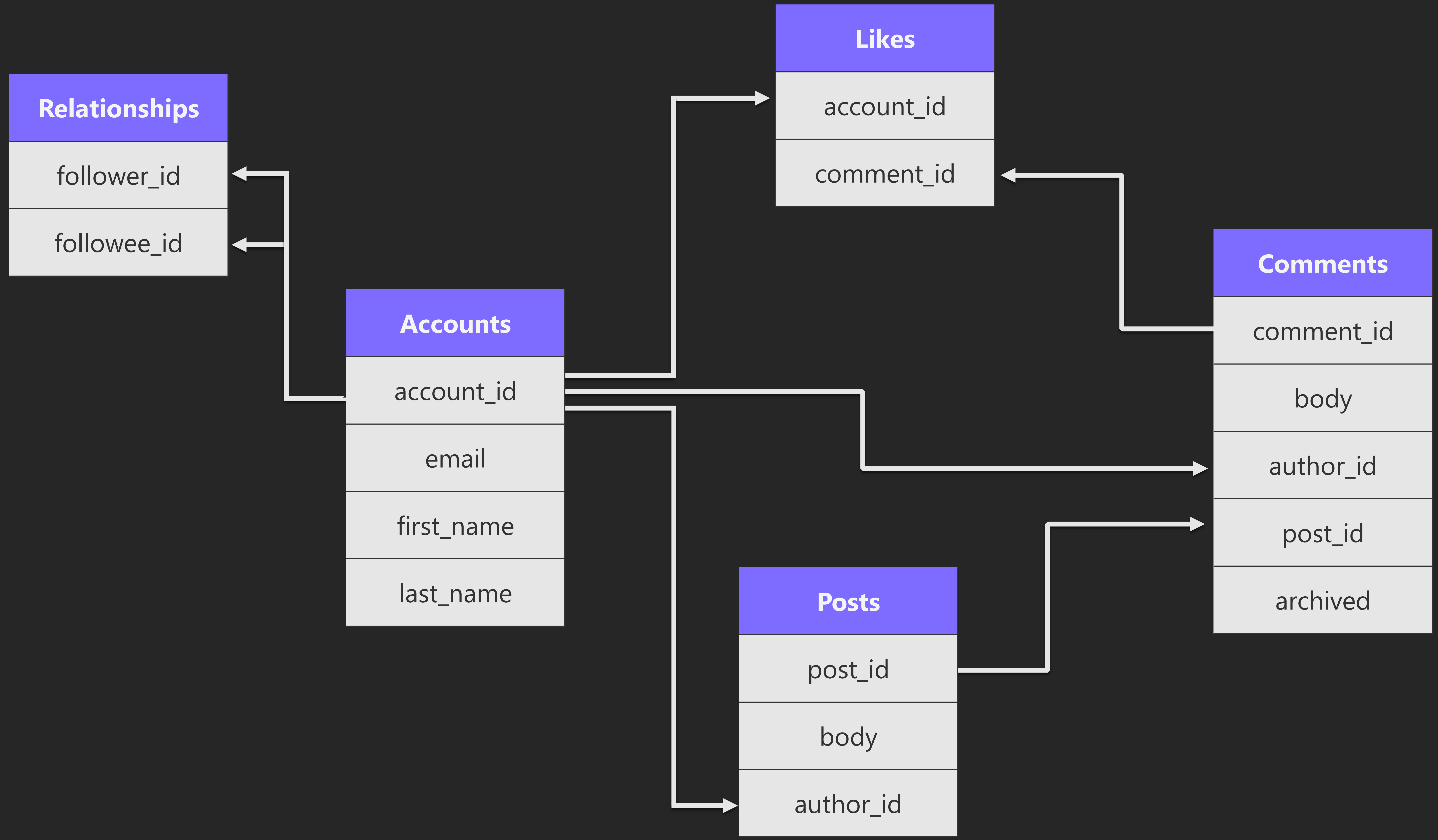
CountryLanguages

language_id	country_id	primary
1	1	True
2	1	False

Languages

language_id	language_name	word_count
1	Spanish	150,000
2	English	600,000

Exercise – Convert Schema



Exercise – Linked List

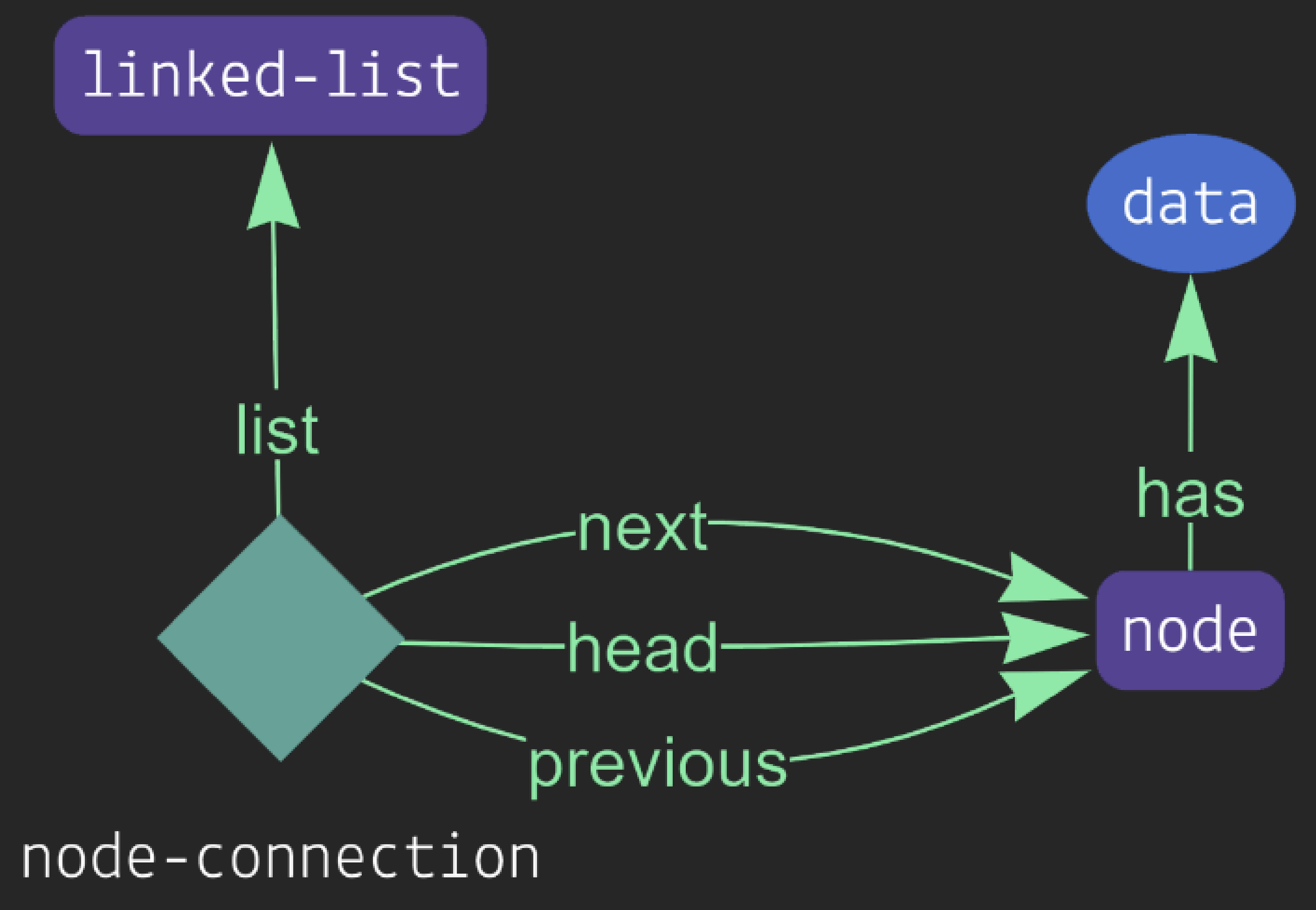
Objectives

1. Implement a linked list using the given schema
2. Populate the linked list with the given data
3. Write queries to answer following questions:
    1. Who is in queue before Alexis?
    2. Who is in queue after the person after Bob?
    3. Who is the last person in the queue?

Data

Bob –> Sam -> Jessica -> Eric -> Alexis

Schema



Exercise – Find Friends-of-Friends

Objectives

1. Implement the given schema
2. Populate the knowledge base with the given data
3. Write a query to find the friends of Brandon's friends
4. Create a rule to infer two people may be acquaintances based on mutual friends

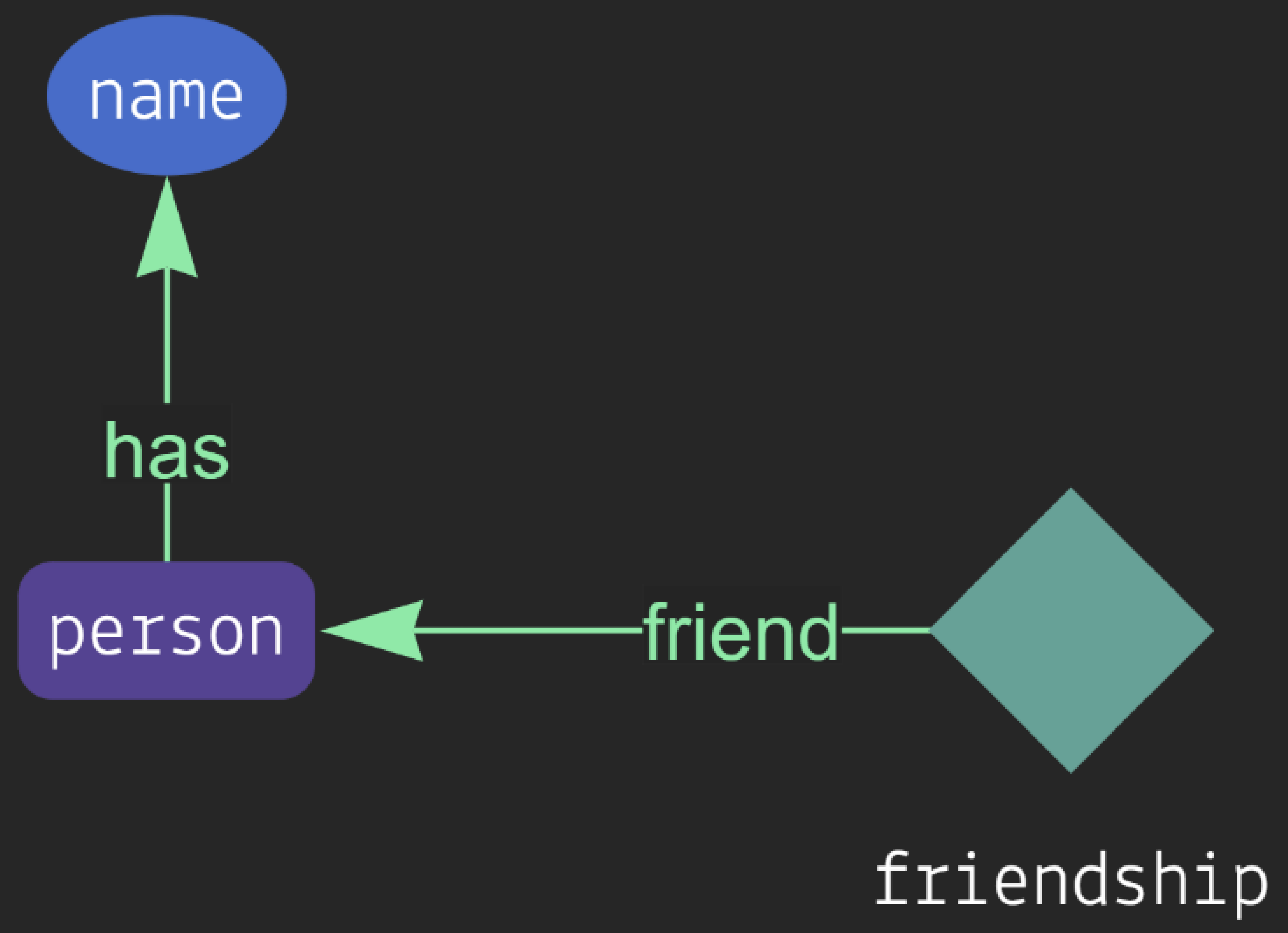
Data

Brandon's Friends:
- Travis
- Mark
- Bob
- Melissa

Mark's Friends:
- Bob

Melissa's Friends:
- Louise
- Alice

Schema



Exercise – Find Skilled Colleagues

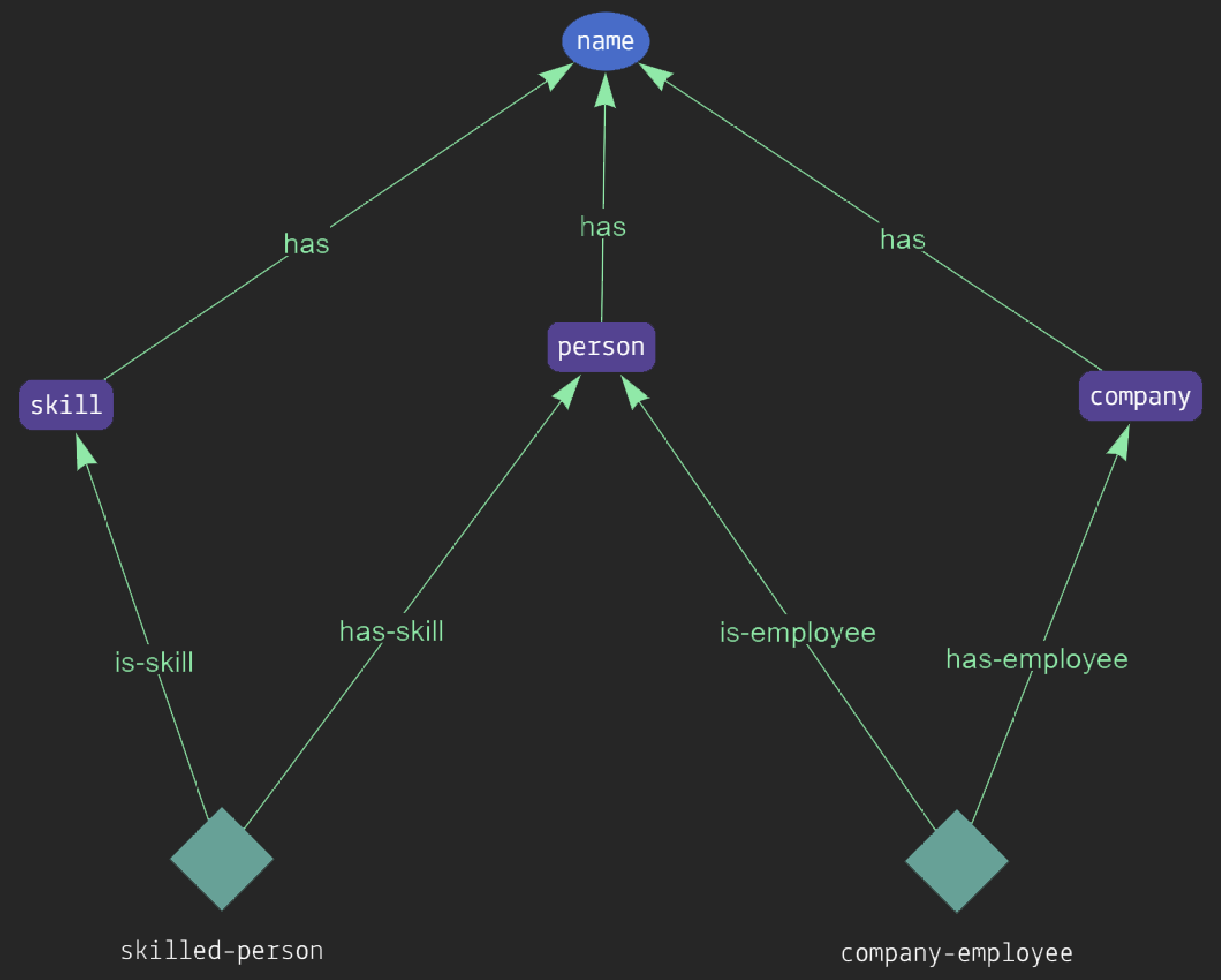
Objectives

1. Implement the given schema
2. Populate the knowledge base with the given data
3. Write a query to find everyone who is good at Kotlin and Java
4. Write a query to find everyone who has no experience with C#
5. Extend the graph to add the ability to assign skill level (weight)
   1. Update Eric to have a skill level of 3 in current skills
   2. Update Larry to have a skill level of 2 in current skills
   3. Update Sergey to have a skill level of 4 in current skills
   4. Write a query to find everyone with a skill level greater than 2 in C#

Data

Company: Google
   Employees:
      Eric
         Skills: Python, C#
      Larry
         Skills: C#, Kotlin, Java
      Sergey
         Skills: Kotlin, Java

Schema



Questions/Comments?

Resources:

- https://grakn.ai
- https://neo4j.com