Physical Data Modeling

Physical Data Modeling

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

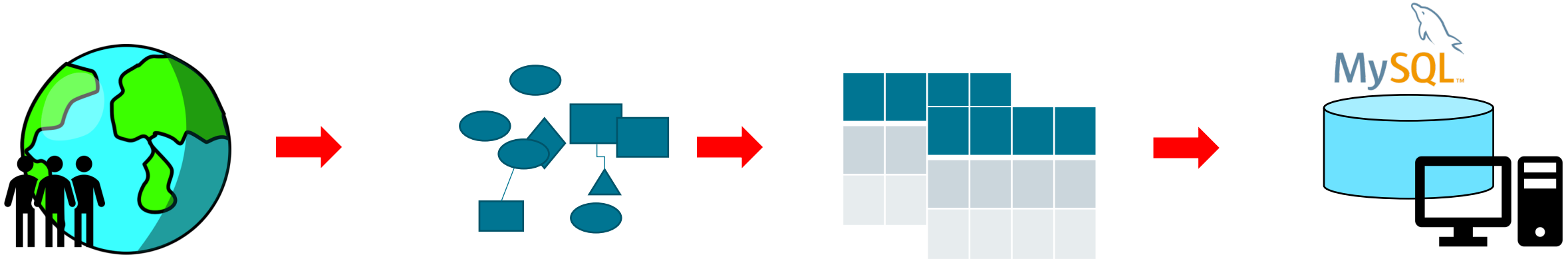
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Define and give examples of physical data models.
- Explain the differences between conceptual and physical data modeling.

Data Modeling Process



Real World

ER Model (Diagrams)

Relational Model (Tables)

DBMS

Conceptual

Physical

The process of data modeling for creating a database

What Is a Physical Data Model?

A model used by an actual database system.

- IMS: Hierarchical model.
- **MySQL: Relational model (our focus).**
- Objectivity/DB : Object-oriented.
- PostgreSQL: Object-relational model.
- Sedna: XML model.
- MongoDB: Document (JSON) model.
- Redis: Key-value model.
- Neo4j: Graph model.

Example database systems

Conceptual vs. Physical: ER Model vs. Relational Model

- Both are used to model data.
- ER model
  - Has more concepts: Entities, relationships.
  - Close to how we view the real world, and thus good for conceptualizing.
  - Does not have computation/operations on its structures.
- Relational model
  - Just one concept: relation.
  - Good for efficient storage/manipulations on computers.
  - Equipped with algebra/operations to define computation on data.

Physical Data Models

Physical Data Modeling

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

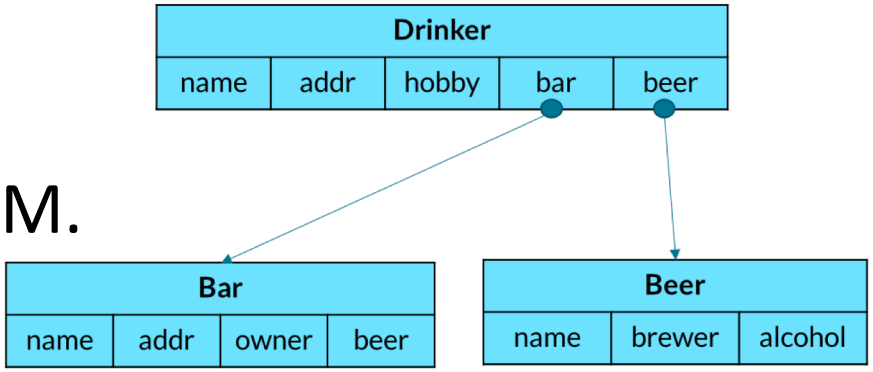
By the end of this video, you will be able to:

- Identify major physical data models.
- Explain the evolution of physical data models.

Physical Data Models

- Pre-relational
  - (1960s) Hierarchical Model.
  - (1960s) Network Model.
- **(1970s) Relational Model -- which is our focus!**
- Post-relational
  - (1980s) Object-Oriented.
  - (1980s) Object-Relational Model.
  - (1990s) Document Model.
  - (1990s) Key-Value Model.
  - (2000s) Graph Model.

Hierarchical Model



- First database model. Created in 1960s at IBM.
- Records connected in hierarchies.
- Data accessed by traversing the hierarchies.

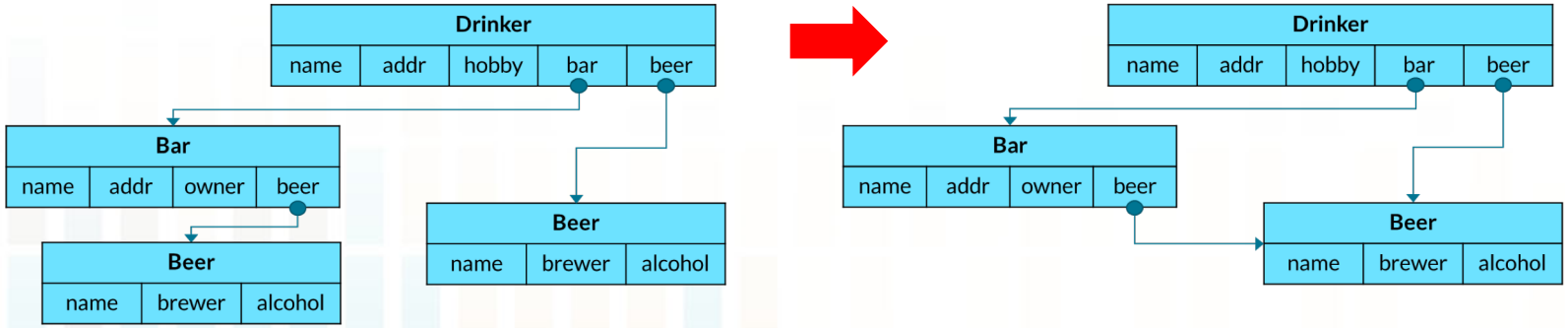
Example hierarchical model

- Example: **IBM IMS**, 1968 ~ Present!
- IBM Information Management System. Still active used!
- *"IMS is used by many of the top Fortune 1000 companies worldwide. Collectively these companies process more than **50 billion transactions per day** in IMS."* IBM Information Management System (IMS), 2017.
- Twitter account: @IBM_IMS, @IMS_at_50.

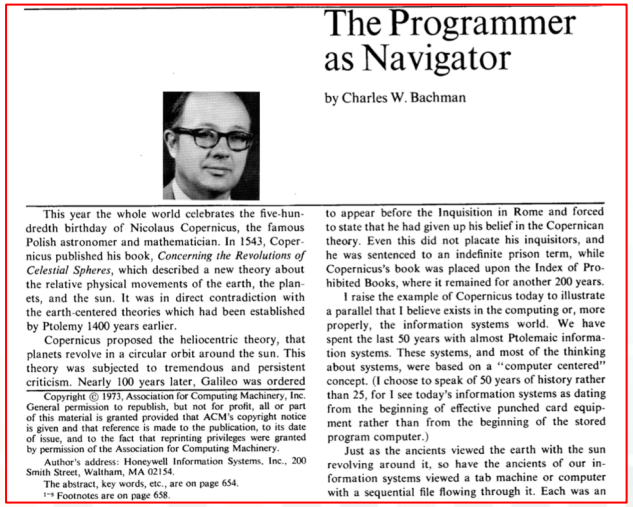
IMS DB and TM Twitter account. Retrieved from https://twitter.com/IMS_at_50

Network Model

- Inspired by hierarchical model, addressing many restrictions.
- Extending tree-like hierarchies to more general networks.
- Data accessed by navigation the networks, from any node.
- Invented by Charles Backman, winning 1973 Turing Award.
- Standardized by CODASYL (Conference on Data Systems Languages Consortium) 1969.



From hierarchical model to network model



The Programmer as Navigator

by Charles W. Bachman

This year the whole world celebrates the five-hundredth birthday of Nicolaus Copernicus, the famous Polish astronomer and mathematician. In 1543, Copernicus published his book, *Concerning the Revolutions of Celestial Spheres*, which described a new theory about the relative physical movements of the earth, the planets, and the sun. It was in direct contradiction with the earth-centered theories which had been established by Ptolemy 1400 years earlier.

Copernicus proposed the heliocentric theory, that planets revolve in a circular orbit around the sun. This theory was subjected to tremendous and persistent criticism. Nearly 100 years later, Galileo was ordered to appear before the Inquisition in Rome and forced to state that he had given up his belief in the Copernican theory. Even this did not placate his inquisitors, and he was sentenced to an indefinite prison term, while Copernicus's book was placed upon the Index of Prohibited Books, where it remained for another 200 years.

I raise the example of Copernicus today to illustrate a parallel that I believe exists in the computing or, more properly, the information systems world. We have spent the last 50 years with almost Ptolemaic information systems. These systems, and most of the thinking about systems, were based on a "computer centered" concept. (I choose to speak of 50 years of history rather than 25, for I see today's information systems as dating from the beginning of effective punched card equipment rather than from the beginning of the stored program computer.)

Just as the ancients viewed the earth with the sun revolving around it, so have the ancients of our information systems viewed a tab machine or computer with a sequential file flowing through it. Each was an

Author's address: Honeywell Information Systems, Inc., 200 Smith Street, Waltham, MA 02154.

The abstract, key words, etc., are on page 654.

[1-5] Footnotes are on page 658.

The programmer as navigator (Bachman, 1973)

Relational Model

- By Edgar Codd at IBM Labs, 1969, winning 1981 Turing Award.
- Motivation: Network model encodes access paths- fragile, inefficient.
- Records stored in relations, without fixed inter-connections.
- Data accessed by computing over relations.
- Most popular: E.g., MySQL, PostgreSQL.

Drinkers				
name	addr	hobby	bar	beer

Bars			
name	addr	owner	beer

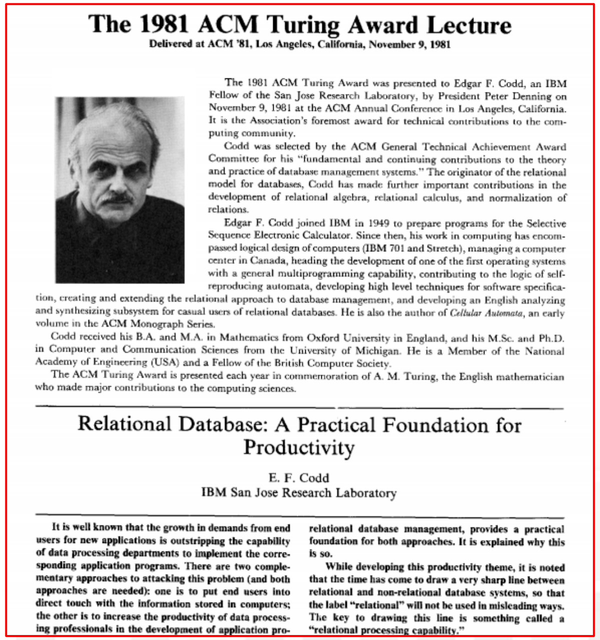
Beers		
name	brewer	alcohol

Frequents	
drinker	bar

Example relational model

- σ (Drinkers)
- Driners $\bowtie$ Bars

Example computation on relations



The 1981 ACM Turing Award Lecture

Delivered at ACM '81, Los Angeles, California, November 9, 1981

The 1981 ACM Turing Award was presented to Edgar F. Codd, an IBM Fellow of the San Jose Research Laboratory, by President Peter Denning on November 9, 1981 at the ACM Annual Conference in Los Angeles, California. It is the Association's foremost award for technical contributions to the computing community.

Codd was selected by the ACM General Technical Achievement Award Committee for his "fundamental and continuing contributions to the theory and practice of database management systems." The originator of the relational model for databases, Codd has made further important contributions in the development of relational algebra, relational calculus, and normalization of relations.

Edgar F. Codd joined IBM in 1949 to prepare programs for the Selective Sequence Electronic Calculator. Since then, his work in computing has encompassed logical design of computers (IBM 701 and Stretch), managing a computer center in Canada, heading the development of one of the first operating systems with a general multiprogramming capability, contributing to the logic of self-reproducing automata, developing high level techniques for software specification, creating and extending the relational approach to database management, and developing an English analyzing and synthesizing subsystem for casual users of relational databases. He is also the author of *Cellular Automata*, an early volume in the ACM Monograph Series.

Codd received his B.A. and M.A. in Mathematics from Oxford University in England, and his M.Sc. and Ph.D. in Computer and Communication Sciences from the University of Michigan. He is a Member of the National Academy of Engineering (USA) and a Fellow of the British Computer Society.

The ACM Turing Award is presented each year in commemoration of A. M. Turing, the English mathematician who made major contributions to the computing sciences.

Relational Database: A Practical Foundation for Productivity

E. F. Codd
IBM San Jose Research Laboratory

It is well known that the growth in demands from end users for new applications is outstripping the capability of data processing departments to implement the corresponding application programs. There are two complementary approaches to attacking this problem (and both approaches are needed): one is to put end users into direct touch with the information stored in computers; the other is to increase the productivity of data processing professionals in the development of application pro- relational database management, provides a practical foundation for both approaches. It is explained why this is so.

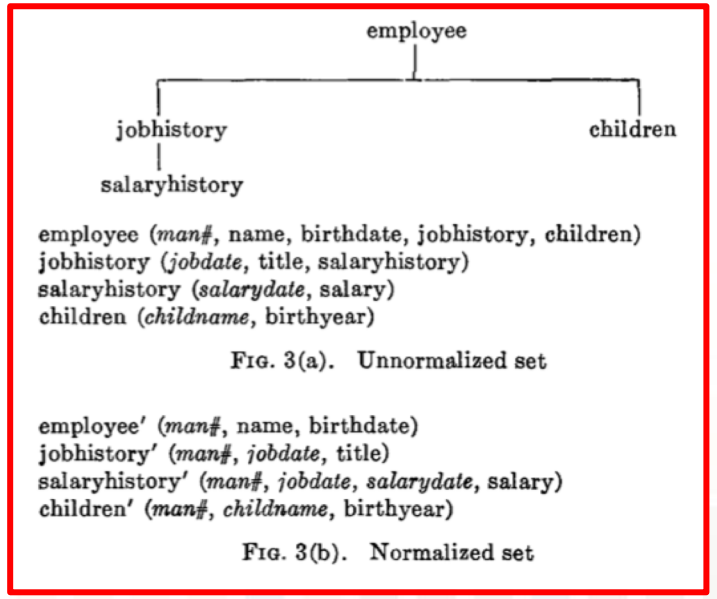
While developing this productivity theme, it is noted that the time has come to draw a very sharp line between relational and non-relational database systems, so that the label "relational" will not be used in misleading ways. The key to drawing this line is something called a "relational processing capability."

Codd's 1982 ACM Turing Award Lecture (Codd, 1982)

Post-Relational Models: Two Driving Forces

- **Meeting programming paradigms**
  - Driven by the ”impedance mismatch” with object-oriented programming.
  - (1980s) Object-Oriented.
  - (1980s) Object-Relational Model.

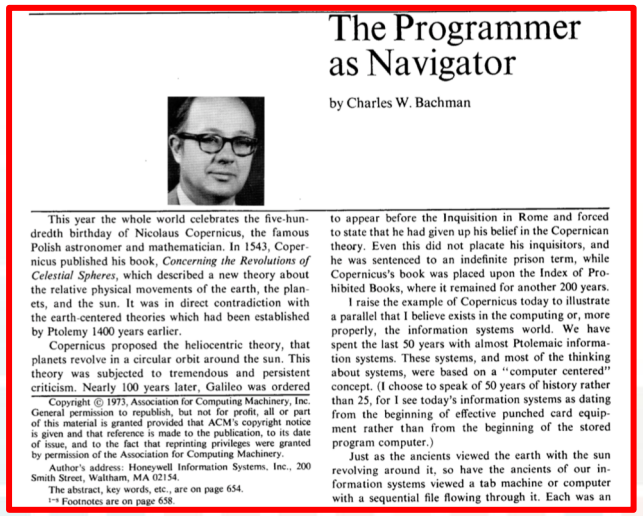
- **Dealing with data in various new settings**
  - Driven by applications beyond enterprise data management.
  - (1990s) Document Model.
  - (1990s) Key-Value Model.
  - (2000s) Graph Model.



Example enterprise data management scenario from Codd’s 1970 relational model paper (Codd 1972)

Food for Thought

*While Bachman touted the triumph of programmers for their **ability and freedom to navigate**, Codd was concerned of the **burden and fragility of the same**. What do you think?*



The Programmer as Navigator

by Charles W. Bachman

This year the whole world celebrates the five-hundredth birthday of Nicolaus Copernicus, the famous Polish astronomer and mathematician. In 1543, Copernicus published his book, *Concerning the Revolutions of Celestial Spheres*, which described a new theory about the relative physical movements of the earth, the planets, and the sun. It was in direct contradiction with the earth-centered theories which had been established by Ptolemy 1400 years earlier.

Copernicus proposed the heliocentric theory, that planets revolve in a circular orbit around the sun. This theory was subjected to tremendous and persistent criticism. Nearly 100 years later, Galileo was ordered to appear before the Inquisition in Rome and forced to state that he had given up his belief in the Copernican theory. Even this did not placate his inquisitors, and he was sentenced to an indefinite prison term, while Copernicus's book was placed upon the Index of Prohibited Books, where it remained for another 200 years.

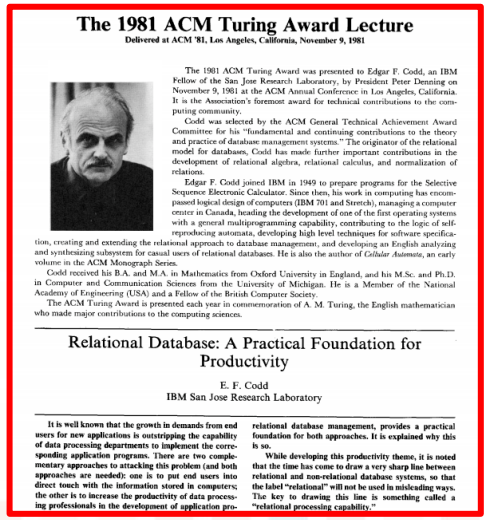
I raise the example of Copernicus today to illustrate a parallel that I believe exists in the computing or, more properly, the information systems world. We have spent the last 50 years with almost Ptolemaic information systems. These systems, and most of the thinking about systems, were based on a "computer centered" concept. (I choose to speak of 50 years of history rather than 25, for I see today's information systems as dating from the beginning of effective punched card equipment rather than from the beginning of the stored program computer.)

Just as the ancients viewed the earth with the sun revolving around it, so have the ancients of our information systems viewed a tab machine or computer with a sequential file flowing through it. Each was an

Author's address: Honeywell Information Systems, Inc., 200 Smith Street, Waltham, MA 02154.

The abstract, key words, etc., are on page 654.
[1-9] Footnotes are on page 658.

vs.



The 1981 ACM Turing Award Lecture

Delivered at ACM '81, Los Angeles, California, November 9, 1981

The 1981 ACM Turing Award was presented to Edgar F. Codd, an IBM Fellow of the San Jose Research Laboratory, by President Peter Denning on November 9, 1981 at the ACM Annual Conference in Los Angeles, California. It is the Association's foremost award for technical contributions to the computing community.

Codd was selected by the ACM General Technical Achievement Award Committee for his "fundamental and continuing contributions to the theory and practice of database management systems." The originator of the relational model for databases, Codd has made further important contributions in the development of relational algebra, relational calculus, and normalization of relations.

Edgar F. Codd joined IBM in 1949 to prepare programs for the Selective Sequence Electronic Calculator. Since then, his work in computing has encompassed logical design of computers (IBM 701 and Stretch), managing a computer center in Canada, heading the development of one of the first operating systems with a general multiprogramming capability, contributing to the logic of self-reproducing automata, developing high level techniques for software specification, creating and extending the relational approach to database management, and developing an English analyzing and synthesizing subsystem for casual users of relational databases. He is also the author of *Cellular Automata*, an early volume in the ACM Monograph Series.

Codd received his B.A. and M.A. in Mathematics from Oxford University in England, and his M.Sc. and Ph.D. in Computer and Communication Sciences from the University of Michigan. He is a Member of the National Academy of Engineering (USA) and a Fellow of the British Computer Society.

The ACM Turing Award is presented each year in commemoration of A. M. Turing, the English mathematician who made major contributions to the computing sciences.

Relational Database: A Practical Foundation for Productivity

E. F. Codd
IBM San Jose Research Laboratory

It is well known that the growth in demands from end users for new applications is outstripping the capability of data processing departments to implement the corresponding application programs. There are two complementary approaches to attacking this problem (and both approaches are needed): one is to put end users into direct touch with the information stored in computers; the other is to increase the productivity of data processing professionals in the development of application programs. Relational languages address the former; relational database management, provides a practical foundation for both approaches. It is explained why this is so.

While developing this productivity theme, it is noted that the time has come to draw a very sharp line between relational and non-relational database systems, so that the label "relational" will not be used in misleading ways. The key to drawing this line is something called a "relational processing capability."

References

- IBM Information Management System (IMS), 2017. Retrieved from https://www-01.ibm.com/software/data/ims/index.html.
- Bachman, 1973. Charles W. Bachman, *The Programmer as Navigator.* ACM Turing Award lecture, Communications of the ACM, Volume 16, Issue 11, 1973, pp. 653–658.
- Codd, 1982 . E. F. Codd: Relational Database: A Practical Foundation for Productivity. Commun. ACM 25(2): 109-117 (1982).
- Codd 1972. E. F. Codd: A Relational Model of Data for Large Shared Data Banks. Commun. ACM 13(6): 377-387 (1970).

Relational Model

Physical Data Modeling

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Define what the relational model is.
- Identify the elements of a relation.
- Describe how schemas are specified and shown in a DBMS.
- Define schema and instance and explain their differences.

An Example of a Relation

Attributes, Fields, Columns

Students

Tuples, Records, Rows

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	Bio	1997-02-01
3	Peter Pan	Econ	1998-10-01
4	Mickey Mouse	CS	1995-04-01

An example relation

Domains

- Each attribute has a **data type**, called its **domain**.
- Must be **atomic** type-- simple values, no further structure.
- Common types: integer, string, real, …
- The exact domains supported depend on the system used.
- Examples:
  - SQLite
    - text, integer, real, blob.
  - MySQL
    - char, varchar, int, float, date, time, year, …

SQLite

MySQL

Schemas of Relations and Databases

- **Schema of a RELATION**
  - Relation name, attribute names, and their domains.
    - Students(id: string, name: string, major: enum(’cs', ’ece', ’ss', ’music’), birthdate: date)
  - May omit domains in notation if they are clear.
    - Students(id, name, major, birthdate)

- **Schema of a DATABASE**
  - A set of relation schemas
    - Students(id, name, major, birthdate)
    - Professors(id, name, dept, course)
    - Courses(number, title, credit)
    - … …

Constraints: Part of a Schema

- Key constraint
  - Unique: Attributes must be unique among tuples in the table.
  - Primary key: Attributes are the primary key of the table.
- Null constraint
  - Not NULL: Attributes cannot be missing/unspecified.
- Default constraint
  - The default value will be added to records if no other value is specified.

Specifying and Showing Schemas

- Using the SQL language to create tables.
- E.g., MySQL
  - Specifying schema for a table:

```
mysql> CREATE TABLE Students (
    ->        id char(10),
    ->        name varchar(100),
    ->        major enum('CS', 'EE', 'Bio', 'Econ'),
    ->        birthday date
    ->        );
Query OK, 0 rows affected (0.02 sec)
```

Screenshot of a MySQL session for specifying a schema

  - Showing schema for a bale:

```
mysql> DESCRIBE Students;
+----------+-------------------------------+------+-----+---------+-------+
| Field    | Type                          | Null | Key | Default | Extra |
+----------+-------------------------------+------+-----+---------+-------+
| id       | char(10)                      | YES  |     | NULL    |       |
| name     | varchar(100)                  | YES  |     | NULL    |       |
| major    | enum('CS','EE','Bio','Econ') | YES  |     | NULL    |       |
| birthday | date                          | YES  |     | NULL    |       |
+----------+-------------------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Screenshot of a MySQL session for showing a schema

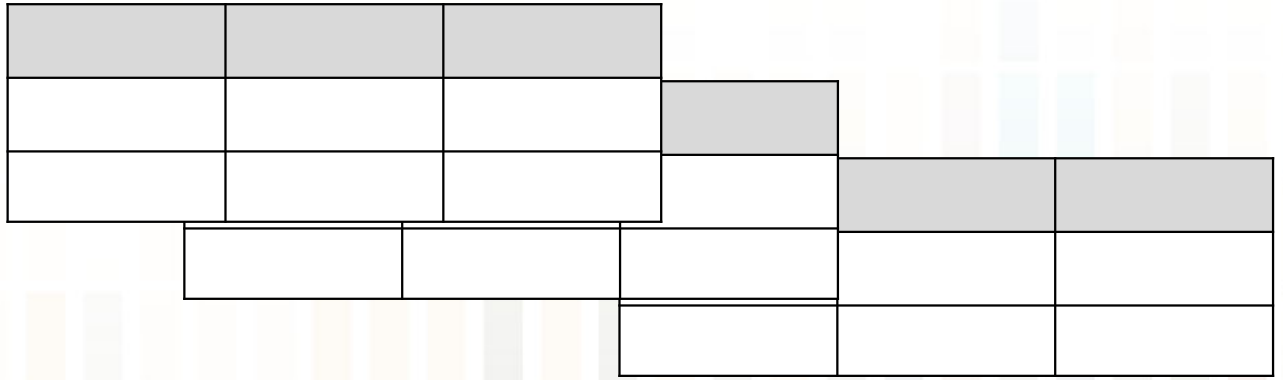
Instances of a Schema

- **Relational schema** $R(A_1, \ldots, A_k)$
- **Instance:** Set of tuples for the relation

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	Bio	1997-02-01
3	Peter Pan	Econ	1998-10-01
4	Mickey Mouse	CS	1995-04-01

An example relation

- **Database schema** $R_{1(\ldots)}, \ldots, R_n(\ldots)$
  Instance = instances of $R_1, \ldots, R_n$.



An example database instance

Updates: Changing Instances

- The database maintains a current database state.
- Updates to instance: Frequent.
  - Add a tuple.
  - Delete a tuple.
  - Modify a tuple.

```
mysql> INSERT INTO Students VALUES ('1', 'Bugs Bunny', 'CS', '2004-11-06');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('2', 'Donald Duck', 'Bio', '1997-02-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('3', 'Peter Pan', 'Econ', '1998-10-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('4', 'Mickey Mouse', 'CS', '1995-04-01');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM Students;
+------+--------------+-------+------------+
| id   | name         | major | birthday   |
+------+--------------+-------+------------+
| 1    | Bugs Bunny   | CS    | 2004-11-06 |
| 2    | Donald Duck  | Bio   | 1997-02-01 |
| 3    | Peter Pan    | Econ  | 1998-10-01 |
| 4    | Mickey Mouse | CS    | 1995-04-01 |
+------+--------------+-------+------------+
4 rows in set (0.00 sec)
```

Screenshot of a MySQL session for updating database instances

Updates: Changing Schemas

- Updates to schema: Infrequent.
  - Add/delete an attribute.
  - Change domains of an attribute.
  - Add/delete a table.
- Think of it as columns vs. rows of a table
  - Rows change much more frequently than columns.

```
mysql> ALTER TABLE Students MODIFY id int;
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Students ADD hobby char(30);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE Students;
+----------+-------------------------------+------+-----+---------+-------+
| Field    | Type                          | Null | Key | Default | Extra |
+----------+-------------------------------+------+-----+---------+-------+
| id       | int(11)                       | YES  |     | NULL    |       |
| name     | varchar(100)                  | YES  |     | NULL    |       |
| major    | enum('CS','EE','Bio','Econ') | YES  |     | NULL    |       |
| birthday | date                          | YES  |     | NULL    |       |
| hobby    | char(30)                      | YES  |     | NULL    |       |
+----------+-------------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM Students;
+------+--------------+-------+------------+-------+
| id   | name         | major | birthday   | hobby |
+------+--------------+-------+------------+-------+
|    1 | Bugs Bunny   | CS    | 2004-11-06 | NULL  |
|    2 | Donald Duck  | Bio   | 1997-02-01 | NULL  |
|    3 | Peter Pan    | Econ  | 1998-10-01 | NULL  |
|    4 | Mickey Mouse | CS    | 1995-04-01 | NULL  |
+------+--------------+-------+------------+-------+
4 rows in set (0.00 sec)
```

Screenshot of a MySQL session for updating database schemas

Food for Thought

Changing the schema of a database could be a painful process. Can you imagine why?

```
mysql> CREATE TABLE Students (
    ->        id char(10),
    ->        name varchar(100),
    ->        major enum('CS', 'EE', 'Bio', 'Econ'),
    ->        birthday date
    ->        );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> INSERT INTO Students VALUES ('1', 'Bugs Bunny', 'CS', '2004-11-06');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('2', 'Donald Duck', 'Bio', '1997-02-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('3', 'Peter Pan', 'Econ', '1998-10-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES ('4', 'Mickey Mouse', 'CS', '1995-04-01');
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> SELECT * FROM Students;
+------+-------------+-------+------------+
| id   | name        | major | birthday   |
+------+-------------+-------+------------+
| 1    | Bugs Bunny  | CS    | 2004-11-06 |
| 2    | Donald Duck | Bio   | 1997-02-01 |
| 3    | Peter Pan   | Econ  | 1998-10-01 |
| 4    | Mickey Mouse| CS    | 1995-04-01 |
+------+-------------+-------+------------+
4 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE Students MODIFY id int;
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Students ADD hobby char(30);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE Students;
+----------+-------------------------------+------+-----+---------+-------+
| Field    | Type                          | Null | Key | Default | Extra |
+----------+-------------------------------+------+-----+---------+-------+
| id       | int(11)                       | YES  |     | NULL    |       |
| name     | varchar(100)                  | YES  |     | NULL    |       |
| major    | enum('CS','EE','Bio','Econ')  | YES  |     | NULL    |       |
| birthday | date                          | YES  |     | NULL    |       |
| hobby    | char(30)                      | YES  |     | NULL    |       |
+----------+-------------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM Students;
+------+-------------+-------+------------+-------+
| id   | name        | major | birthday   | hobby |
+------+-------------+-------+------------+-------+
|    1 | Bugs Bunny  | CS    | 2004-11-06 | NULL  |
|    2 | Donald Duck | Bio   | 1997-02-01 | NULL  |
|    3 | Peter Pan   | Econ  | 1998-10-01 | NULL  |
|    4 | Mickey Mouse| CS    | 1995-04-01 | NULL  |
+------+-------------+-------+------------+-------+
4 rows in set (0.00 sec)
```

Screenshot of a MySQL session for updating database schemas

From ER to Relational Model

Physical Data Modeling

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

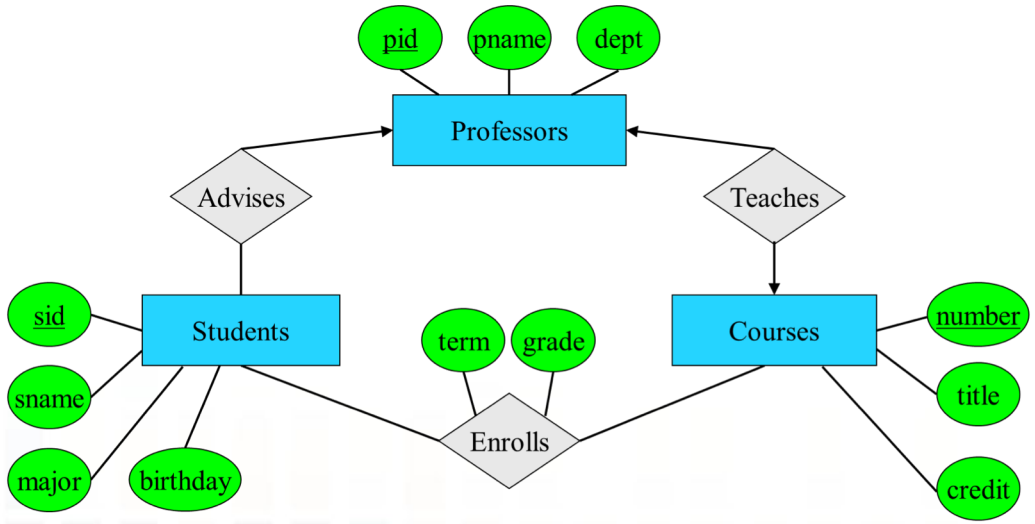
By the end of this video, you will be able to:

- Translate an ER diagram to a relational schema.
- Combine relations to simplify a schema.

Academic World

Note:

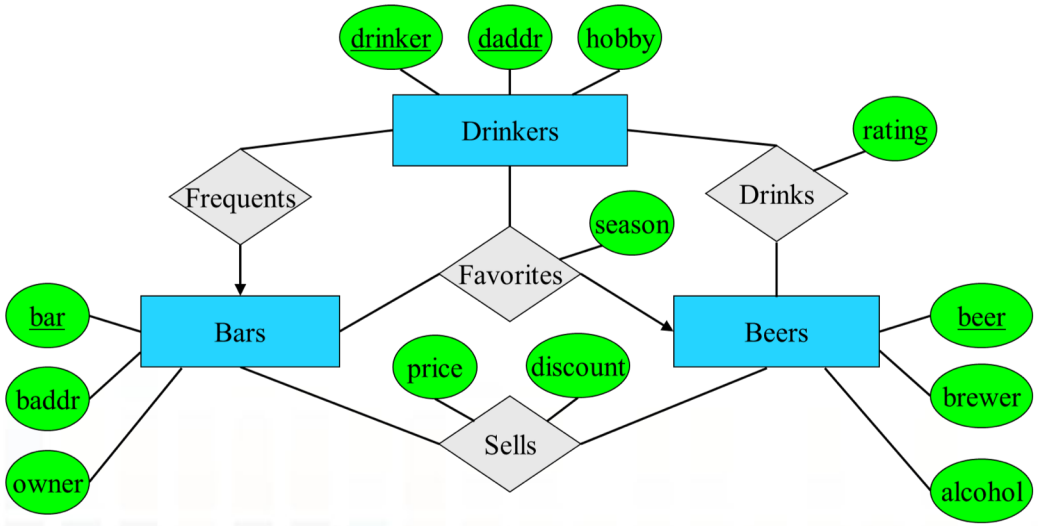
- Changed some attributes (e.g., “name” to “pname” or “sname”) on ER to avoid name clashes.
- This can also be done later when clashes happen during translation too.



Professors(<u>pid</u>, pname, dept, course)
Students(<u>sid</u>, sname, major, birthday, advisor)
Courses(<u>number</u>, title, credit)
Enrolls(<u>sid</u>, <u>number</u>, term, grade)

Translating an ER diagram to a relational schema

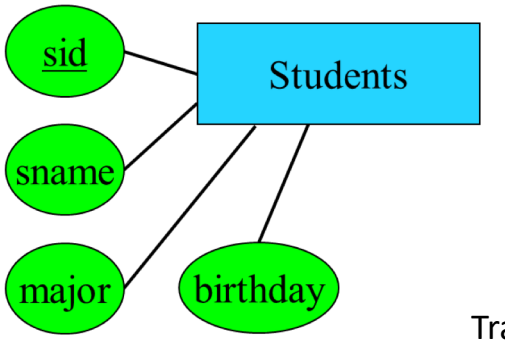
Relational Schema: Friday Night

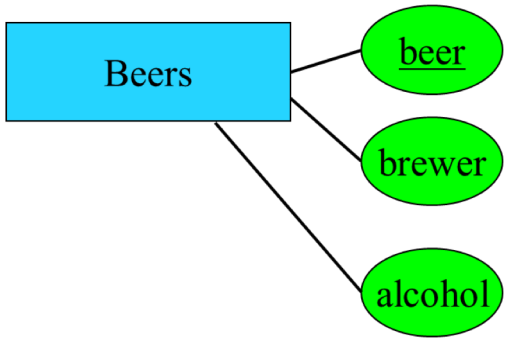


- Drinkers(drinker, daddr, hobby, bar)
- Bars(bar, baddr, owner)
- Beers(beer, brewer, alcohol)
- Sells(bar, beer, price, discount)
- Drinks(drinker, beer, rating)
- Favorites(drinker, bar, beer, season)

Translating an ER diagram to a relational schema

Rule 1: Entity Set → Relation





Translating entity set into relation

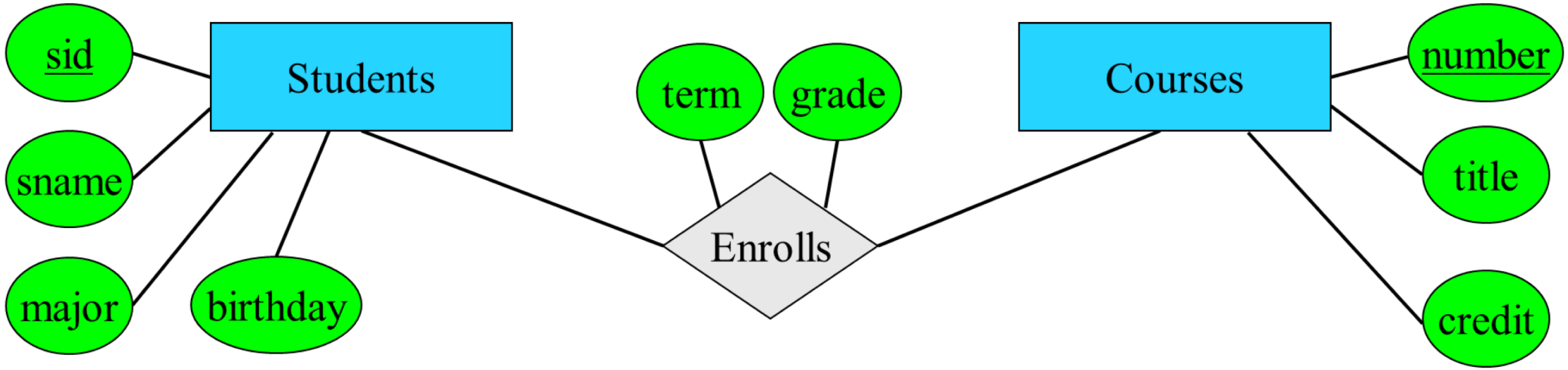
Students(sid, sname, major, birthday)

Beers(beer, brewer, alcohol)

Rule: Translating entity set E to relation R

- Attributes of R = attributes of E.
- Key of R = key of E.

Relationship → Relation: What Attributes?



Enrolls(<u>sid</u>, sname, major, birthday, <u>number</u>, title, credit, term, grade)

Enrolls(<u>sid</u>, <u>number</u>, term, grade)

Translating relationship into relation

Relationship → Relation: What Keys?

Favorites(drinker, bar, beer, season)

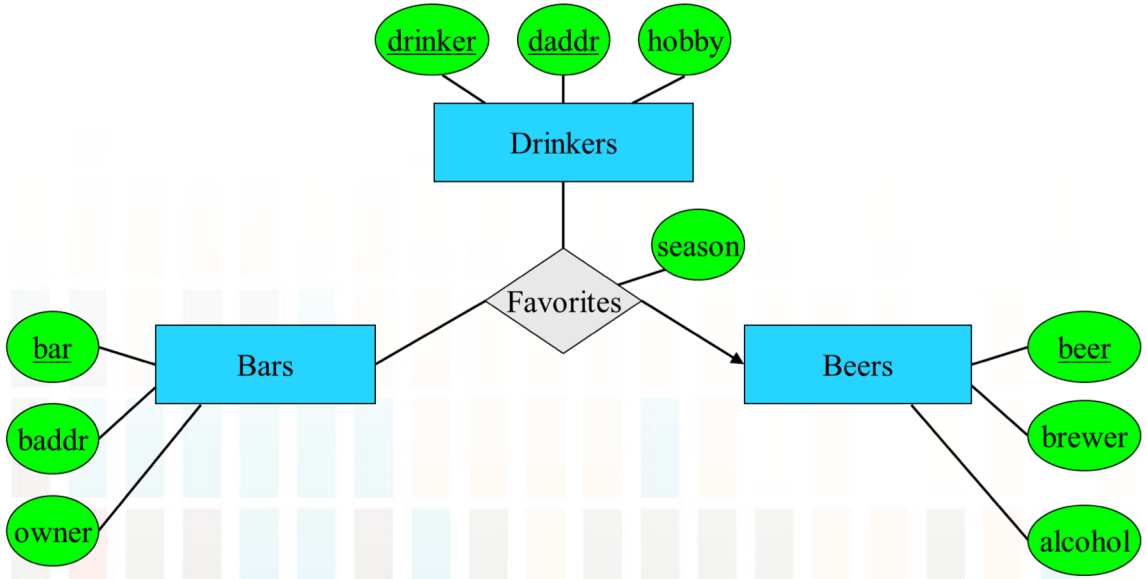
Favorites(drinker, bar, beer, season)

Translating relationship into relation

Rule 2: Relationship → Relation

Rule: Translating relationship X of $E_1 \ldots, E_n$ to relation R

- Attributes of R = key attributes of $E_1, \ldots, E_n$ plus attributes of X
- Key of R = key of $E_1, \ldots, E_n$ except those "arrowed" entities

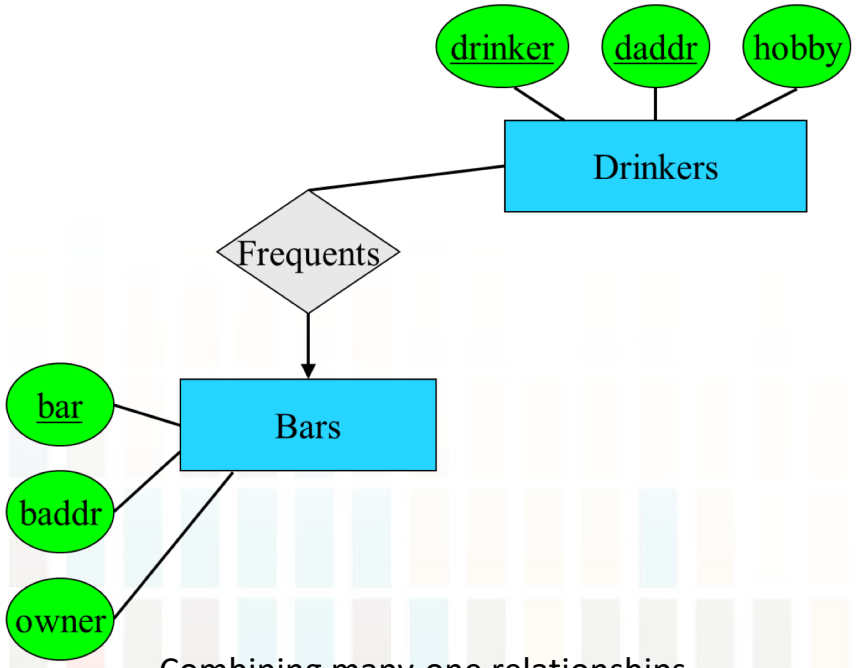


Favorites(drinker, bar, beer, season)

Translating relationship into relation

Rule 3: Combining Many-One Relationships

- Rule: Combine the relation of a many-one relationship with the relation of the “many”-side entity set.



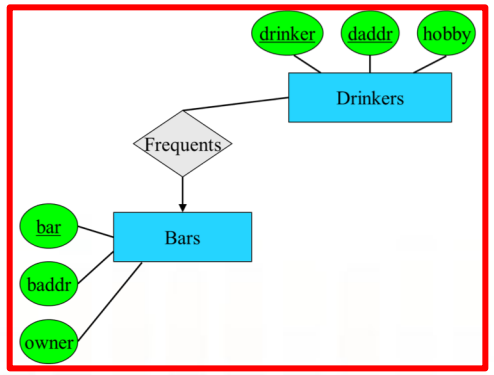
Combining many-one relationships

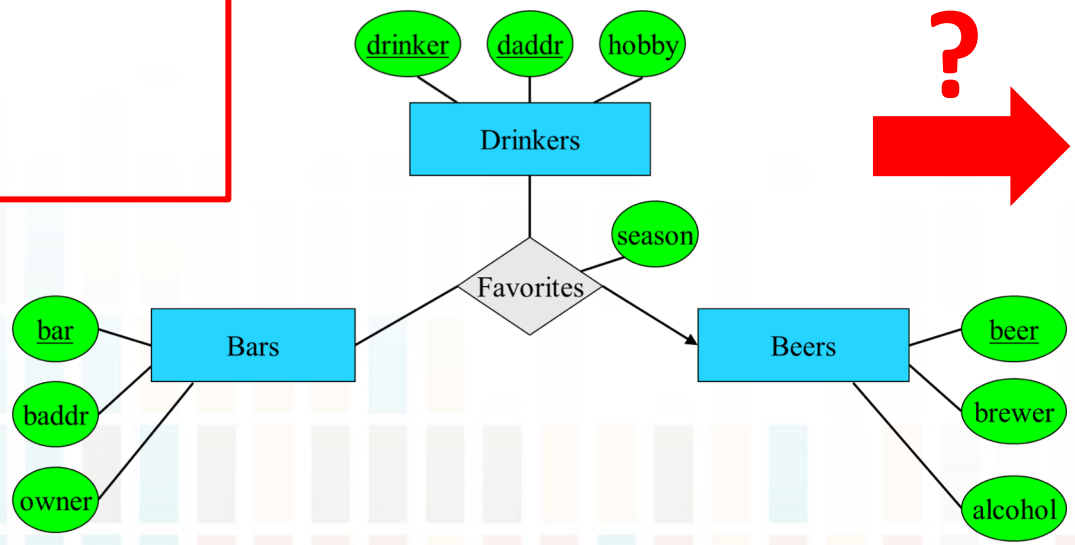
- Drinkers(drinker, daddr, hobby)
- Frequents(drinker, bar)
- Bars(bar, baddr, owner)

- Drinkers(drinker, daddr, hobby, bar)
- Bars(name, baddr, owner)

Food for Thought

We can combine a binary many-one relationship. Can we similarly combine a multiway many-many-one relationship? Say, merge Favorites to Drinkers? How about many-one-one?





?

Drinkers(<u>drinker</u>, daddr, hobby,

bar, beer, season)

Translating Subclasses

Physical Data Modeling

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

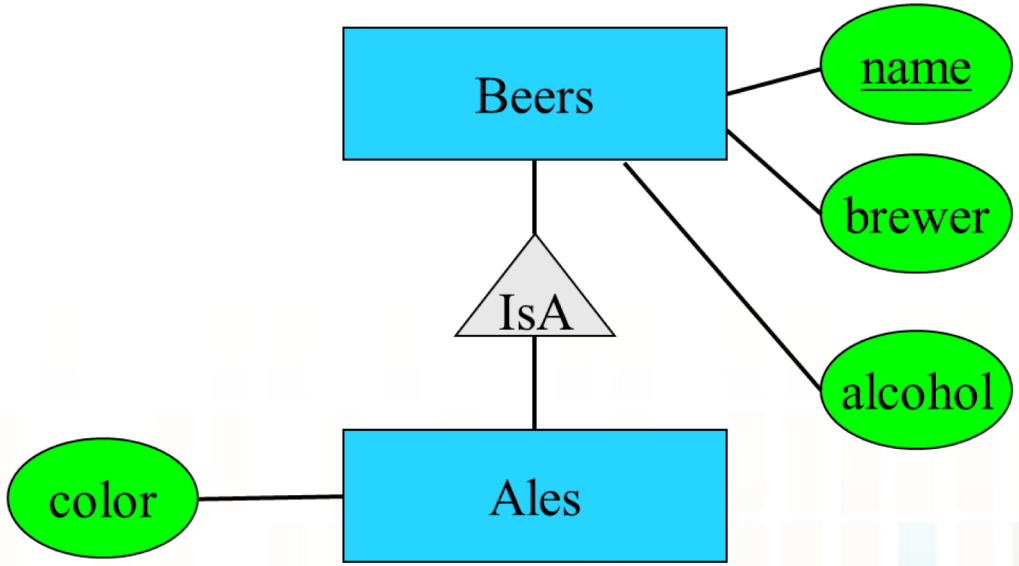
Learning Objectives

By the end of this video, you will be able to:

- Translate subclasses in an ER diagram to a relational schema.
- Identify the options for such translation and their differences.

Expressing Subclasses in Relations

- Relational model is not object-oriented.
- Object features were motivation for new models.
  - Object-oriented.
  - Object-relational.



Example subclass

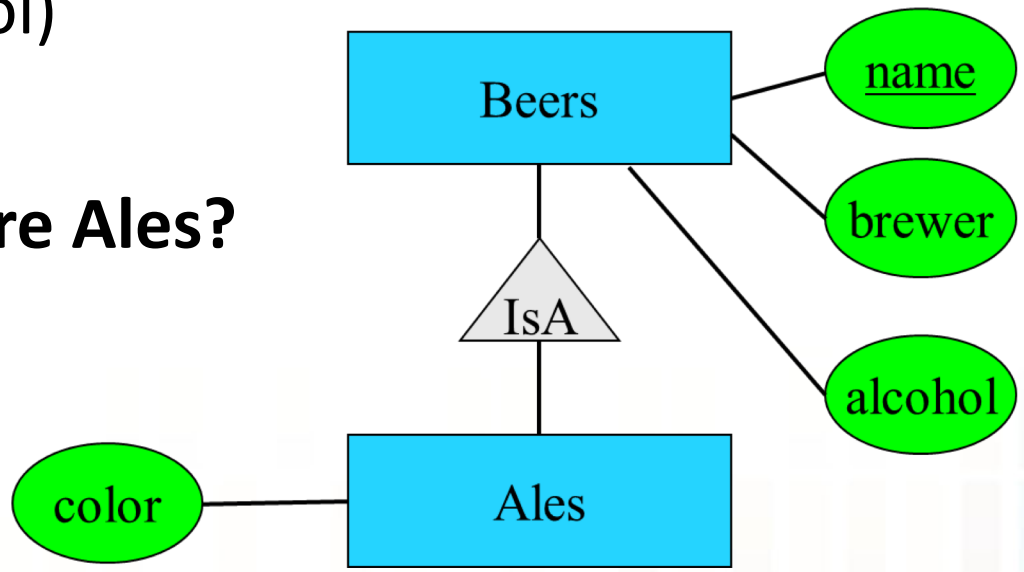
How to Express a Subclass Entity-Set?

- We need a relation for Beers. This is normal.

  Beers(<u>name</u>, brewer, alcohol)

- But, **where and how do you store Ales?**

  Hint: No, not a refrigerator.

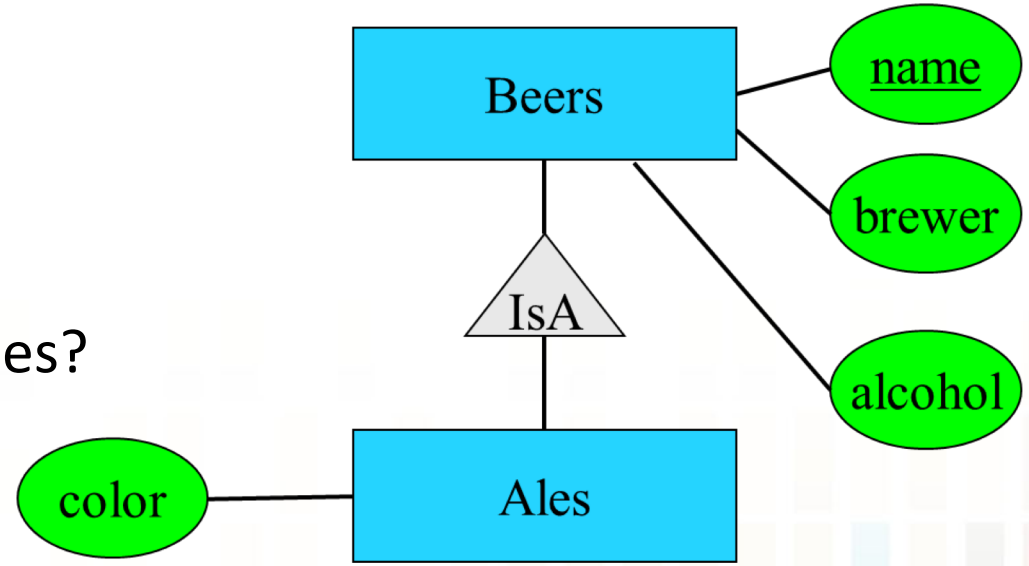


Example subclass

Several Options-- None Perfect

- ER Approach
- Object-oriented approach
- Null-value approach

- Consider them by criteria
  - Performance for common queries?
  - Space need for typical data?

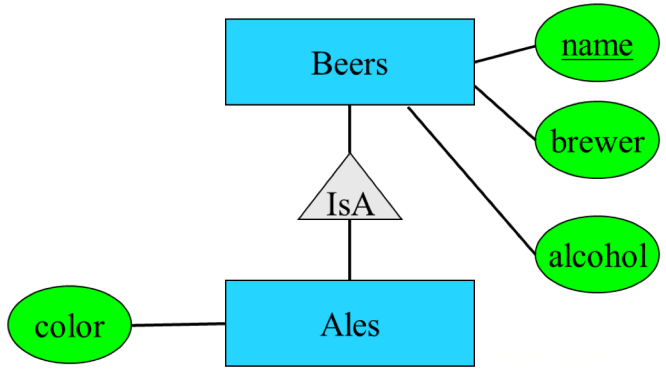


Example subclass

Option 1: The ”ER Approach”

Store Subclass Just Like What ER Suggests

- What does the ER model suggest?
  - Subclass entity-set has its own “additional” attributes.
  - Relate to superclass via IsA.
- Storing subclass as a relation:
  - With only the additional attributes.
  - Link to superclass-relation via its key.
- Ex: Beers(<u>name</u>, brewer, alcohol), Ales(<u>name</u>, color)



Example subclass

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

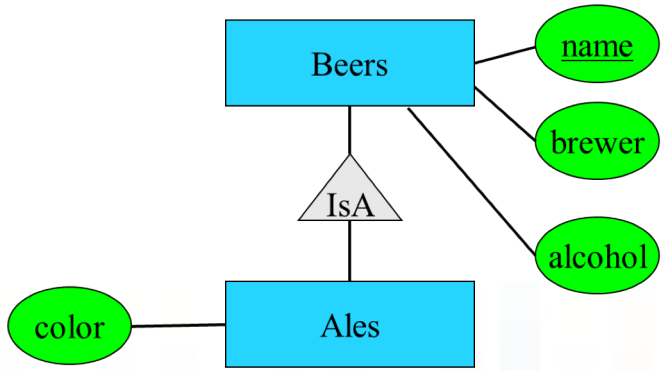
name	color
Goose IPA	Golden
Summer Ale	Dark

Example database

Option 2: The ”OO Approach”

Store Subclass with OO Inheritance

- What would an ”object-oriented approach” suggest?
  - Inheritance: Subclass would inherit attributes from superclass.
- Storing subclass as a relation:
  - With also attributes from the superclass.
- Ex:
  - Beers(<u>name</u>, brewer, alcohol)
  - Ales(<u>name</u>, brewer, alcohol, color)



Example subclass

name	brewer	alcohol
Sam Adams	Boston Beer	4.9

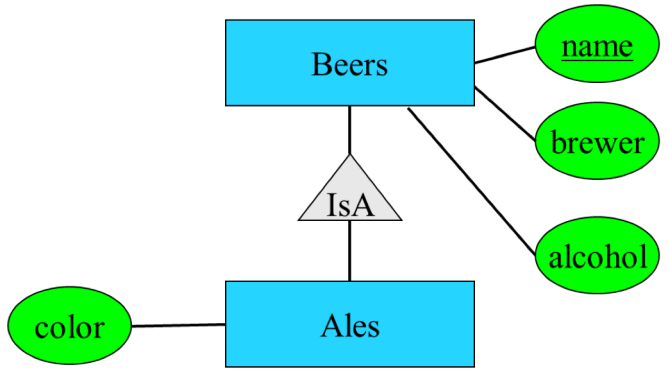
name	brewer	alcohol	color
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

Example database

Option 3: The ”Null-Value Approach”

One Table for Both Super/Subclasses

- Use one relation for both, i.e., “one size fits all”.
- Storing super/subclass in the same relation:
  - With all attributes.
- Use null values to say “N/A”.
- Ex: Beers(<u>name</u>, brewer, alcohol, aleColor)



Example subclass

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

name	color
Goose IPA	Golden
Summer Ale	Dark

ER approach

name	brewer	alcohol
Sam Adams	Boston Beer	4.9

name	brewer	alcohol	color
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

OO approach

name	brewer	alcohol	aleColor
Sam Adams	Boston Beer	4.9	Null
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

Example database

Q: What would work faster?

- Find the colors of ales made by some company, say, Boston Beer.

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

name	color
Goose IPA	Golden
Summer Ale	Dark

1. ER approach

name	brewer	alcohol
Sam Adams	Boston Beer	4.9

name	brewer	alcohol	color
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

2. OO approach

name	brewer	alcohol	aleColor
Sam Adams	Boston Beer	4.9	Null
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

3. Null-value approach

Q: What would work faster?

- Find all beers made by Boston Beer.

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

name	color
Goose IPA	Golden
Summer Ale	Dark

1. ER approach

name	brewer	alcohol
Sam Adams	Boston Beer	4.9

name	brewer	alcohol	color
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

2. OO approach

name	brewer	alcohol	aleColor
Sam Adams	Boston Beer	4.9	Null
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

3. Null-value approach

Food for Thought

Is the Null-Value Approach any good? Help me find some arguments.

Performance-wise? Space-wise?

name	brewer	alcohol	aleColor
Sam Adams	Boston Beer	4.9	Null
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

Null-value approach

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

name	color
Goose IPA	Golden
Summer Ale	Dark

ER approach

name	brewer	alcohol
Sam Adams	Boston Beer	4.9

name	brewer	alcohol	color
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

OO approach

Post-relational: Object-based Modeling

Physical Data Modeling

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

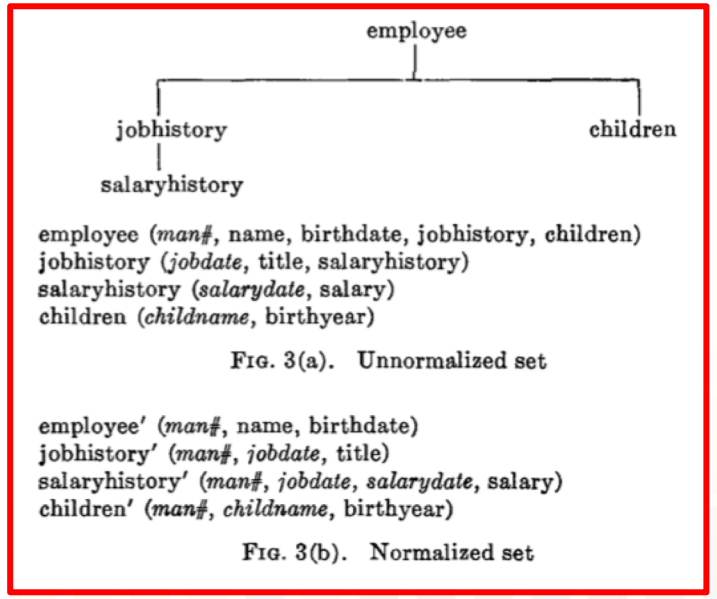
By the end of this video, you will be able to:

- Describe the object-based models after the relational model.
- Identify the motivations behind these models.
- Give examples of such models.

Post-Relational Models: Two Driving Forces

- **Meeting programming paradigms**
  - Driven by the ”impedance mismatch” with object-oriented programming.
  - (1980s) Object-Oriented.
  - (1980s) Object-Relational Model.

- **Dealing with data in various new settings**
  - Driven by applications beyond enterprise data management.
  - (1990s) Document Model.
  - (1990s) Key-Value Model.
  - (2000s) Graph Model.



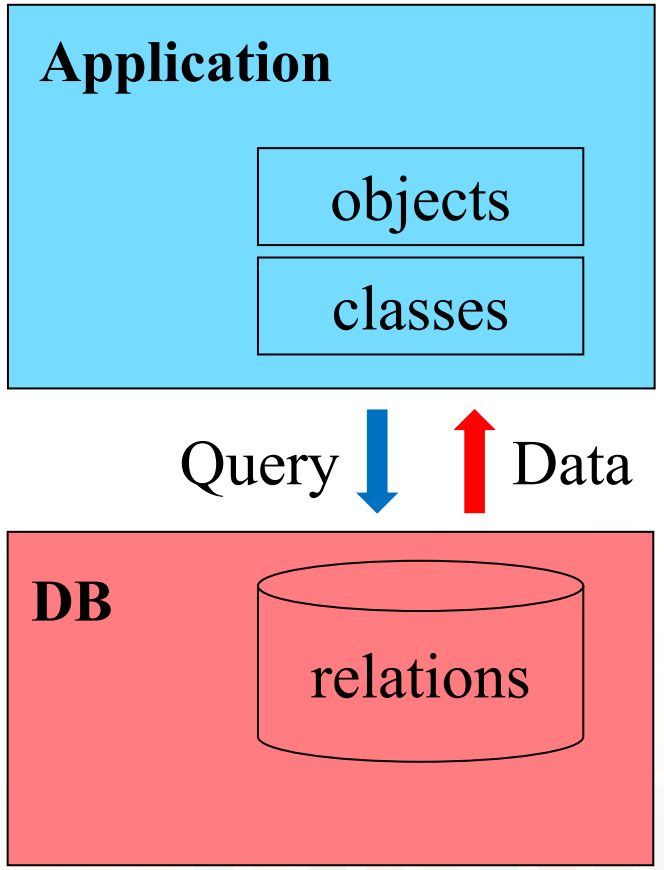
Example enterprise data management scenario from Codd’s 1970 relational model paper (Codd 1972)

Impedance Mismatch: Database vs. Application Programming

Object-relational impedance mismatch --

Object (of class) vs **Tuple** (of relation):

- **Object identifiers**: Object referenced by identifiers (pointers) while tuples by values.
- **Nesting**: Object can be nested to contain objects while tuple contains only simple values as attributes.
- **Methods**: Object has methods while tuple contains only attributes.
- **Inheritance**: Classes can form subclass hierarchy while relations cannot.
- **Encapsulation:** Object can hide internal representation.



Database vs. application programming

Meeting Programming Paradigms → Object-based

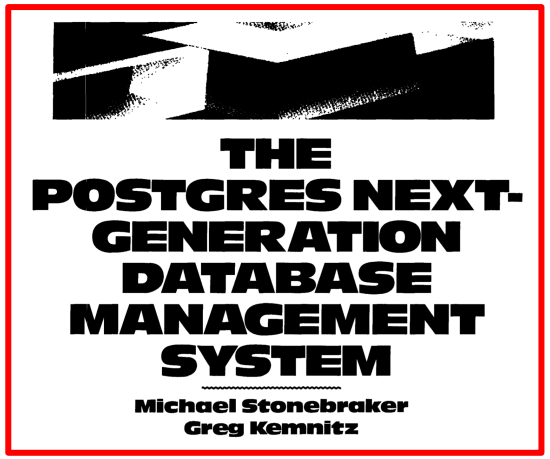
- Object-Oriented Data Model
  - Extending OO programming languages with database functions.
  - Current systems:.

Example OO database systems

- Object-Relational Data Model
  - Extending RDBMS with object features.
  - Current systems: PostgreSQL
    - Most RDBMS supports object features to various extents.

Example object-relational database system

And, this is ANOTHER Turing Award

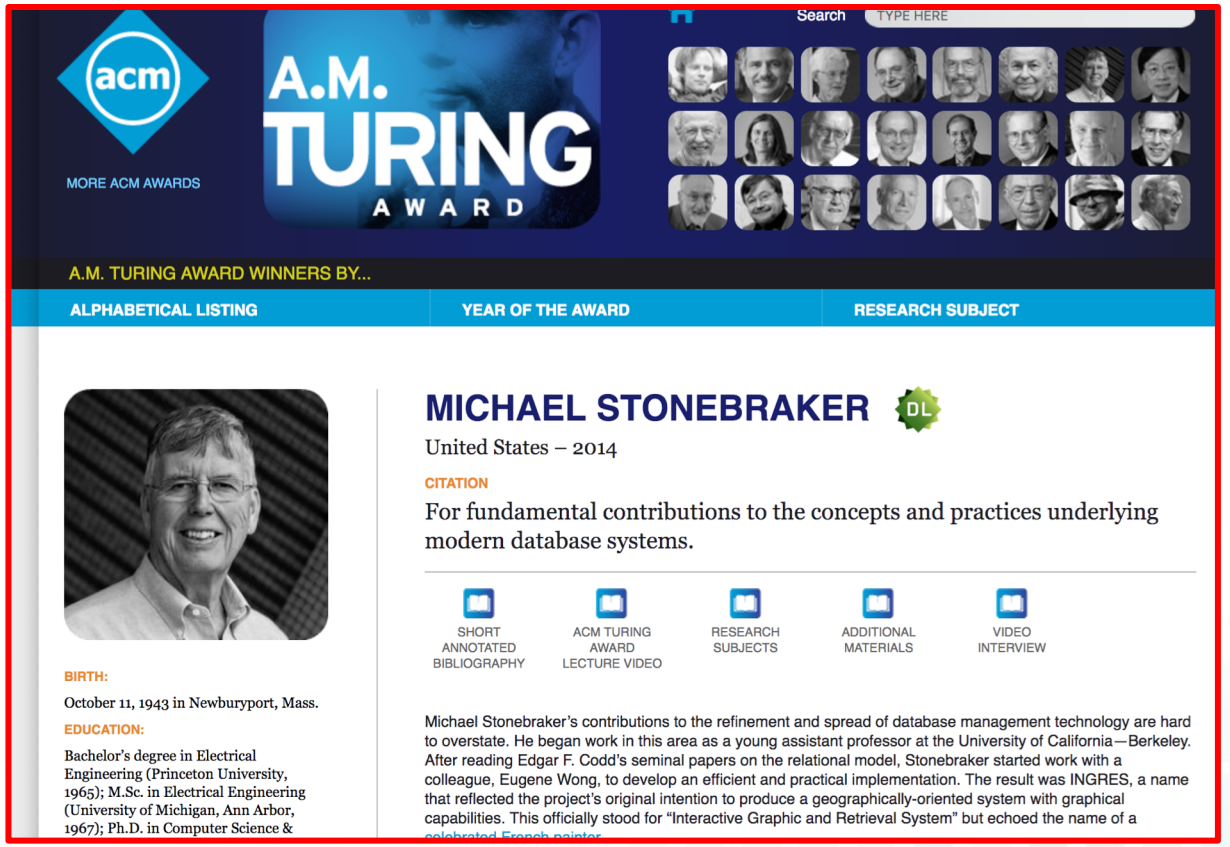


1991

1996

PostgreSQL

From Postgres to PostgreSQL



A.M. Turing Award 2014 citation. Retrieved from http://amturing.acm.org/award_winners/stonebraker_1172121.cfm

“Object-Relational” Example: Table Inheritance

Table inheritance in PostgreSQL

Beers-Ales scenario, in Object-Relational way:

- Superclass

  CREATE TABLE Beers (

  name text,

  brewer text,

  alcohol float)

- Subclass

  CREATE TABLE Ales (

  color char(10)) **INHERITS (Beers);**

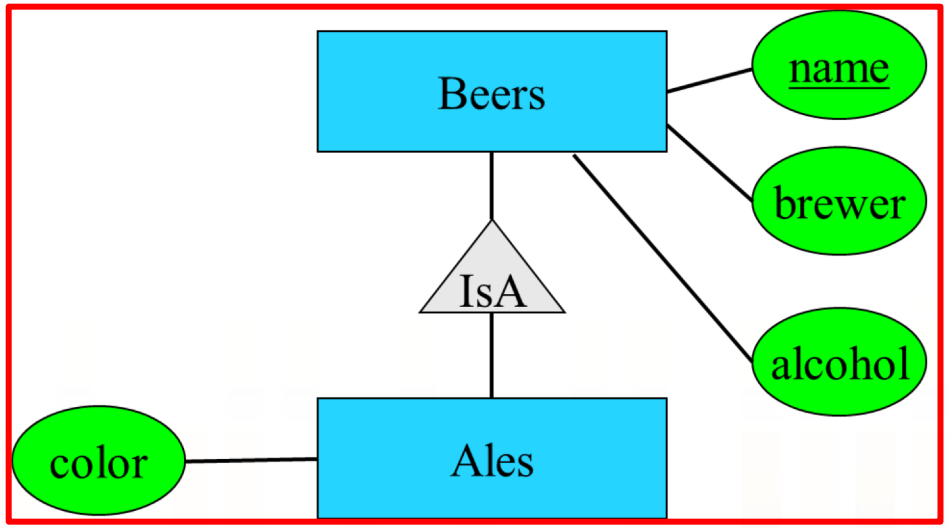
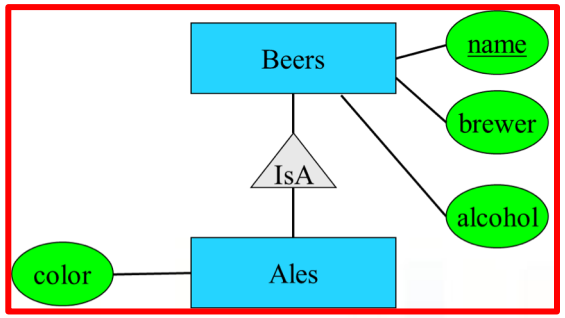


Table Inheritance: Interesting Complications

- When querying about “Beers”, what tables should match?
  - Default is to match not only Beers but also Ales.
- When inserting into “Beers”, what tables should match?
  - Seems natural to only insert into Beers.
- How do I limit matching to only tuples in Beers?
  - You say “ONLY Beers”. E.g.

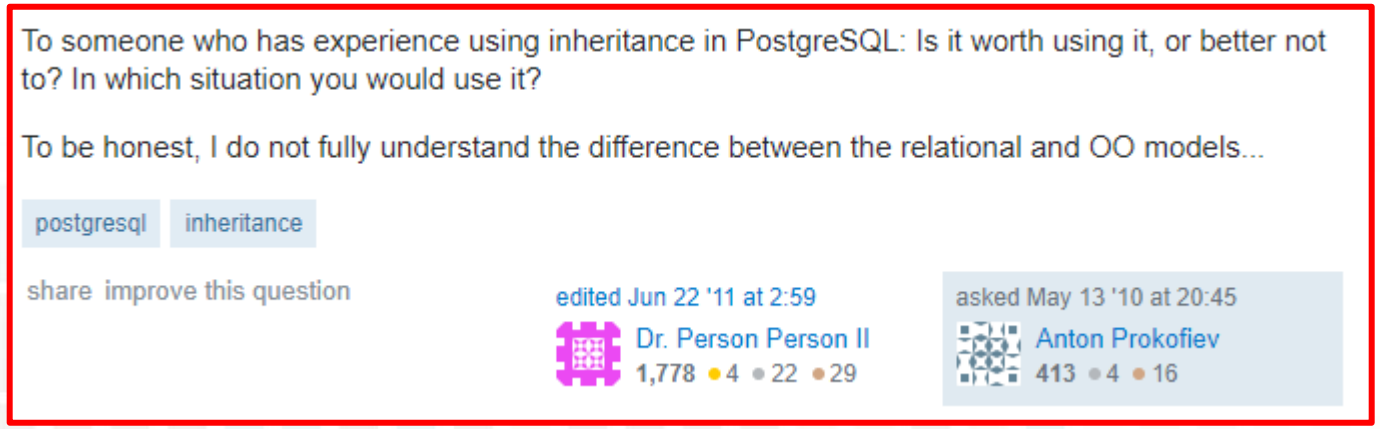
    SELECT name, alcohol FROM **ONLY Beers**



- How to expand matching to not only Beers but also Ales?
  - You say “Beers*”.
- See Inheritance in PostgreSQL Manual.

Food for Thought

On StackOverflow, some suggested against using inheritance in PostgreSQL (and recommended the Null-Value approach). Can you imagine why?



To someone who has experience using inheritance in PostgreSQL: Is it worth using it, or better not to? In which situation you would use it?

To be honest, I do not fully understand the difference between the relational and OO models...

postgresql inheritance

share improve this question

edited Jun 22 '11 at 2:59
Dr. Person Person II
1,778 ● 4 ● 22 ● 29

asked May 13 '10 at 20:45
Anton Prokofiev
413 ● 4 ● 16

Question on table inheritance in PostgreSQL, 2011.
Retrieved from https://stackoverflow.com.

Post-relational: NoSQL Modeling

Physical Data Modeling

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

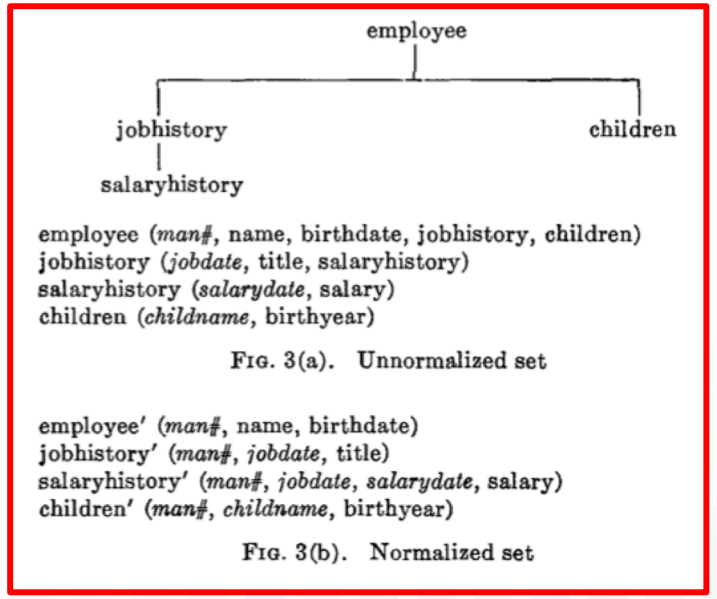
By the end of this video, you will be able to:

- Describe the “NoSQL” data models after the relational model.
- Identify the motivations behind these models.
- Give examples of such models.
- Define what JSON model is and how JSON schema works.

Post-Relational Models: Two Driving Forces

- **Meeting programming paradigms**
  - Driven by the ”impedance mismatch” with object-oriented programming.
  - (1980s) Object-Oriented.
  - (1980s) Object-Relational Model.

- **Dealing with data in various new settings**
  - Driven by applications beyond enterprise data management.
  - (1990s) Document Model.
  - (1990s) Key-Value Model.
  - (2000s) Graph Model.



Example enterprise data management scenario from Codd’s 1970 relational model paper (Codd 1972)

Dealing with Data in Various New Settings → NoSQL

- New kinds of data: Web data, social networks, scientific data.
- New requirements
  - Volume → Scalability
    - Handling extremely large data.
    - Handling extremely many users.
  - Variety → One model may not fit all
    - Handling very simple to very complex data.
- NoSQL databases
  - Originally “non SQL” or “non relational”.
  - Now “not only SQL”.

NoSQL Data Models

- Key-Value Model
  - Berkeley DB, Redis
- Document Model
  - MongoDB, CouchDB
  - JSON is a popular document model.
- Graph Model
  - Neo4j, OrientDB

Key-Value Model

- DB = key-value pairs.
- Key: Any binary sequence given/named by programmers.
- Value: String, or more complex types list, hash, set (as in redis).
- Data model is effectively managed at applications.
- Good for simple data with mostly simple look-up queries.
  - e.g., collection of users, look-up password for logging in.

Example key-value database

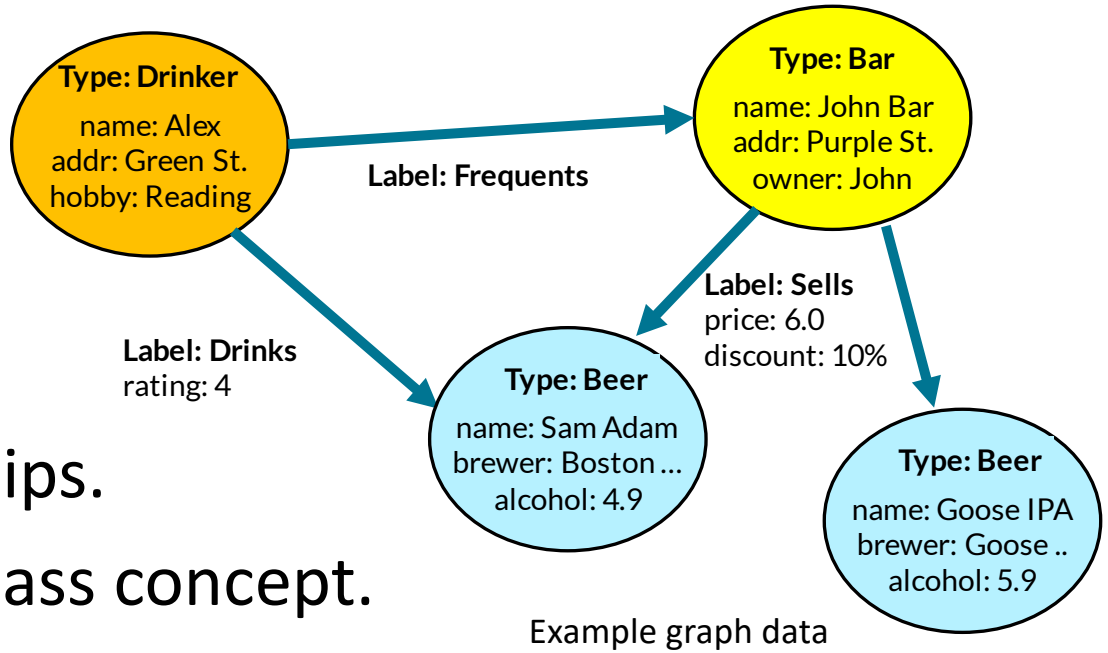
Key	Value
beer:001:name	“Sam Adams”
beer:001:brewer	”Boston Beer”
beer:001:alcohol	4.9
beer:002:name	“Goose IPA”
…	…

Example key-value store

Key	Value
beer:001	{name: “Sam Adams”, brewer:”Boston Beer”, alcohol:4.9}
beer:002	{name:”Goose IPA”, brewer:”Goose Island”, alcohol:5.9}
…	…

Example key-value store, with complex values

Graph Model



Example graph data

- Database = Graph.
- Node = Entities. Edges = Relationships.
- Key concept: Relationship as first class concept.
- *"Think of the ease and beauty of a well-done, normalized entity-relationship diagram: a simple, easy to understand model you can quickly whiteboard with your colleagues and domain experts. A graph is exactly that: a clear model of the domain, focused on the use cases you want to efficiently support."* Neo4j Blog, 2016.
- Network data model coming back?
  - Subtle differences, but similar in spirit.

neo4j

Example graph database

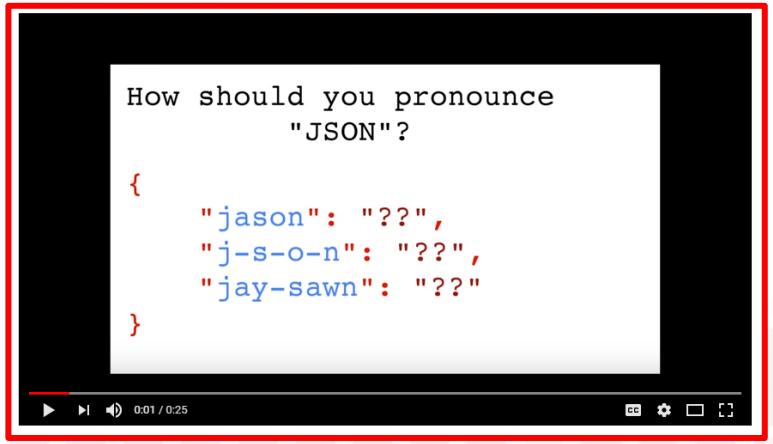
Document Model

- DB = Collections of Documents.
- Document encoded in nested structure, e.g., XML, JSON.
- Documents do not necessarily share the same schema.
  - But you can enforce a schema (or document validation rules).

Example document databases

JSON: JavaScript Object Notation

- Standard for serializing data object in human-readable format.
- Popularized as common format for browser server data exchange.
- Then backend databases start to store data as JSON too.
  - And the idea of document databases emerged.
- Used by different languages/systems beyond JavaScript.
- How to pronounce ”JSON”?



How should you pronounce “JSON”? Retrieved from https://www.youtube.com/watch?v=zhVdWQWKRqM

JSON Documents

- Human readable.
- Value: basic types are strings, numbers, booleans, null.
- Object: { field-value pair }, i.e., set of field-value pairs.
- Array: [value]
- Nesting: Value can be an embed objects or referenced objects (by object id).

```
{
  "_id": "<ObjectId1>",
  "name": "Samuel Adams",
  "brewer": {
    "name": "Boston Beer Company",
    "location": "Boston, Massachusetts"
  },
  "alcohol": 4.9,
  "type": "larger",
  "year introduced": 1984,
  "variants": [
    "<ObjectId2>",
    "<ObjectId3>"
  ]
}
```

```
{
  "_id": "<ObjectId2>",
  "name": "Samuel Adams Light",
  "brewer": {
    "name": "Boston Beer Company",
    "location": "Boston, Massachusetts"
  },
  "alcohol": 3.2,
  "type": "larger",
  "year introduced": 1993
}
```

JSON documents (for Beers Collection)

JSON Schema

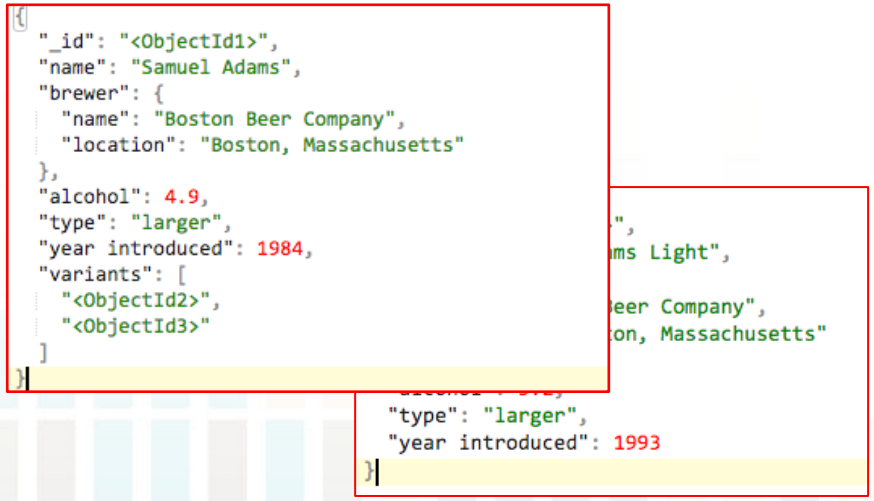
- We can mix different kinds of documents in a collection.
- We can also specify the structure of JSON data– by a JSON schema.
- In JSON format. Human readable
- Define structures
  - Set of properties (fields).
  - Type for each property.
  - Constraint for each property.

JSON schema

```
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "brewer": {
      "type": "object",
      "properties": {}
    },
    "alcohol": {
      "type": "number",
      "minimum": 0,
      "maximum": 100
    },
    "type": {
      "type": "string"
    },
    "year introduced": {
      "type": "number"
    },
    "variants": {
      "type": "array",
      "items": {
        "type": "objectId",
        "minItems": 1,
        "uniqueItems": true
      }
    }
  },
  "required": [
    "name",
    "brewer",
    "alcohol"
  ]
}
```

Food for Thought

*How do you compare the **document model** to the **relational model**? What are major different concepts?*



vs.

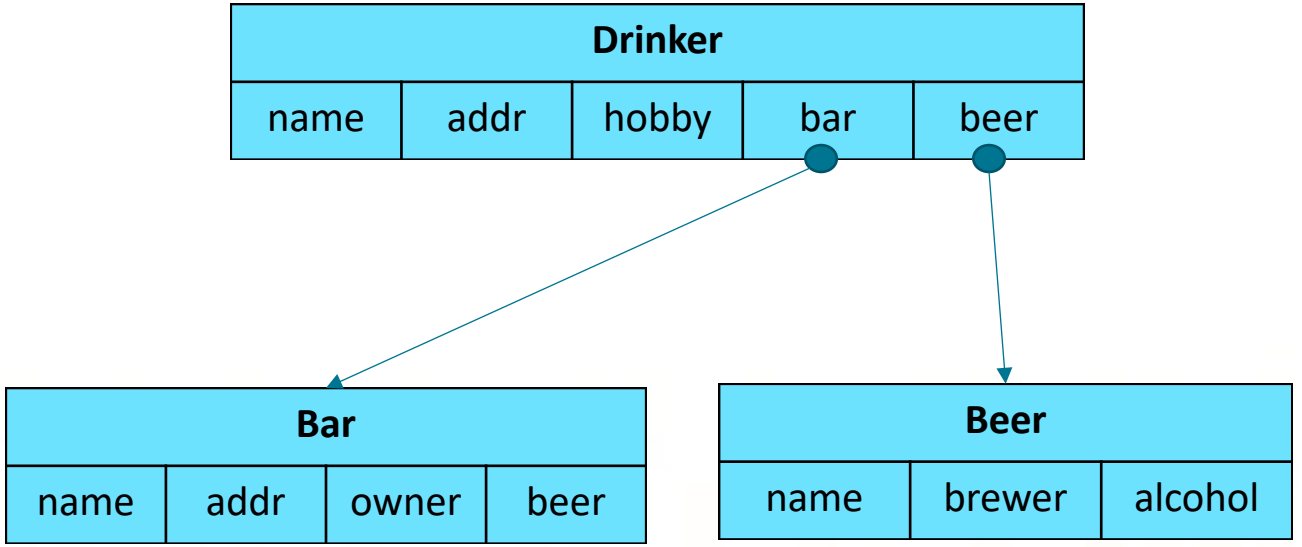
name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

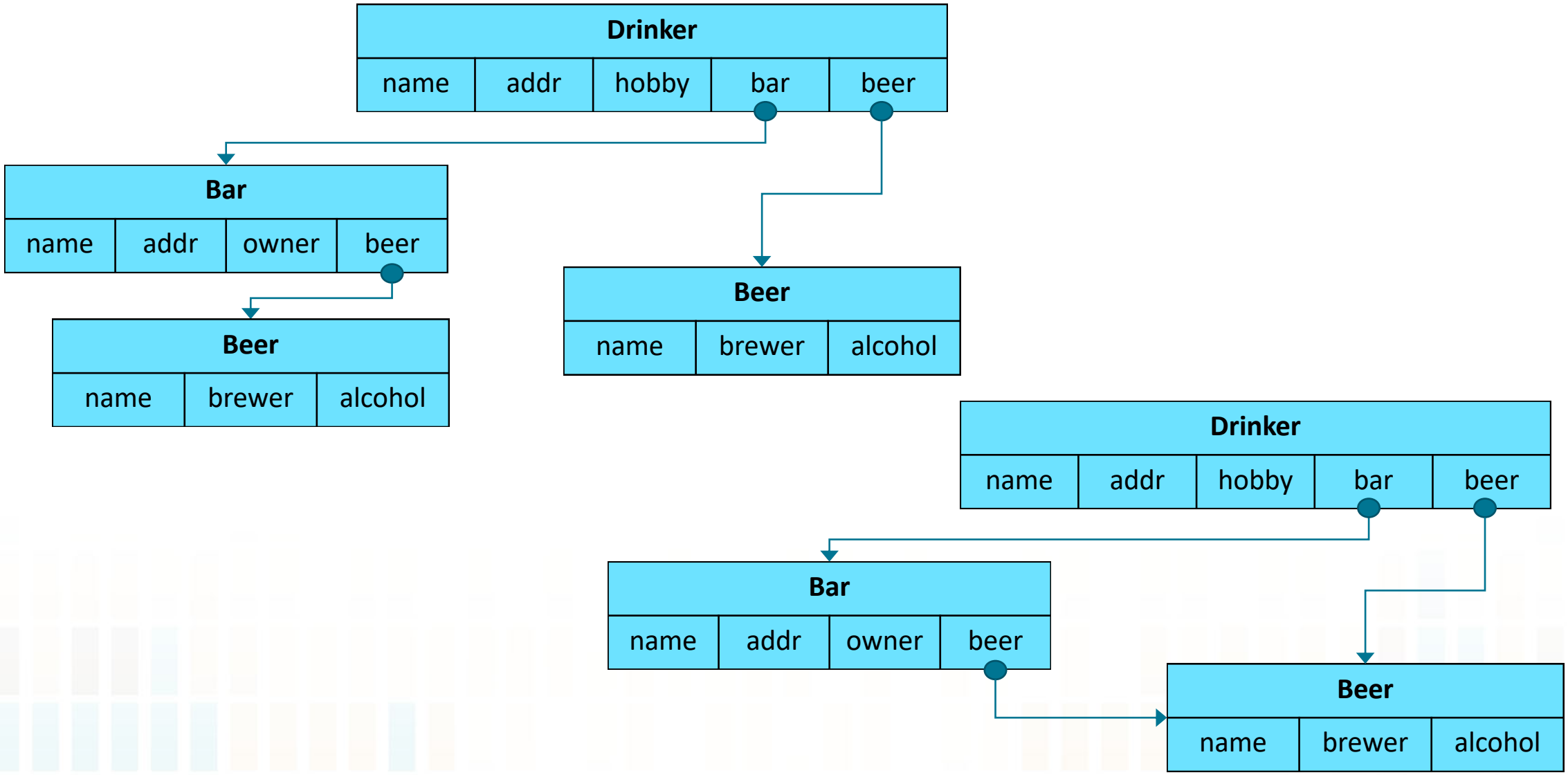
References

- *Neo4j Blog, 2016*. RDBMS & Graphs: Relational vs. Graph Data Modeling. Retrieved from https://neo4j.com/blog/rdbms-vs-graph-data-modeling/.

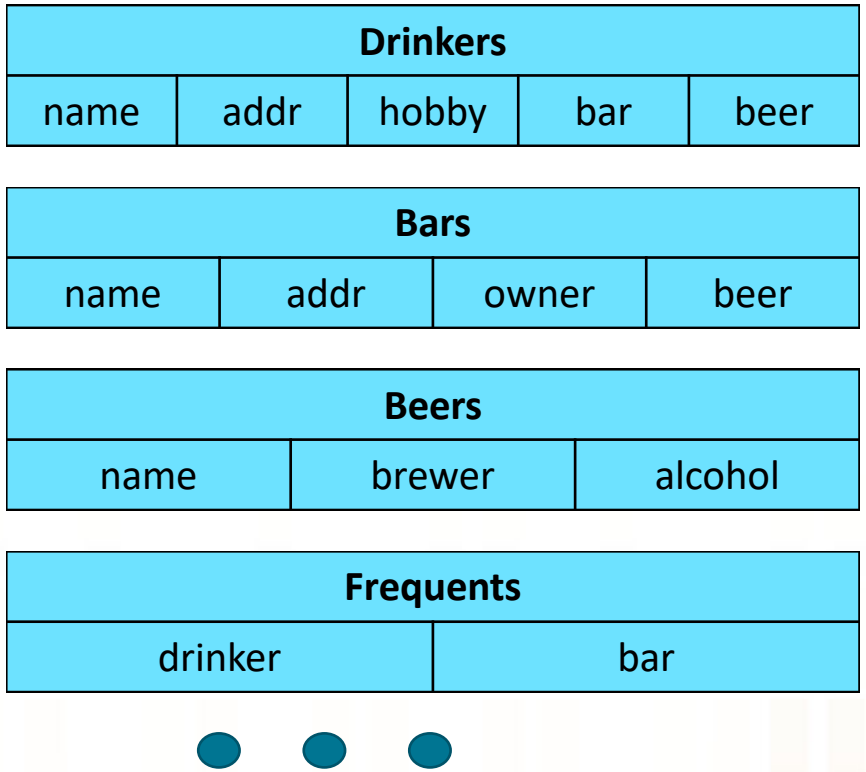
The End

Figure


Drinker
name
addr
hobby
bar
beer
Bar
name
addr
owner
beer
Beer
name
brewer
alcohol

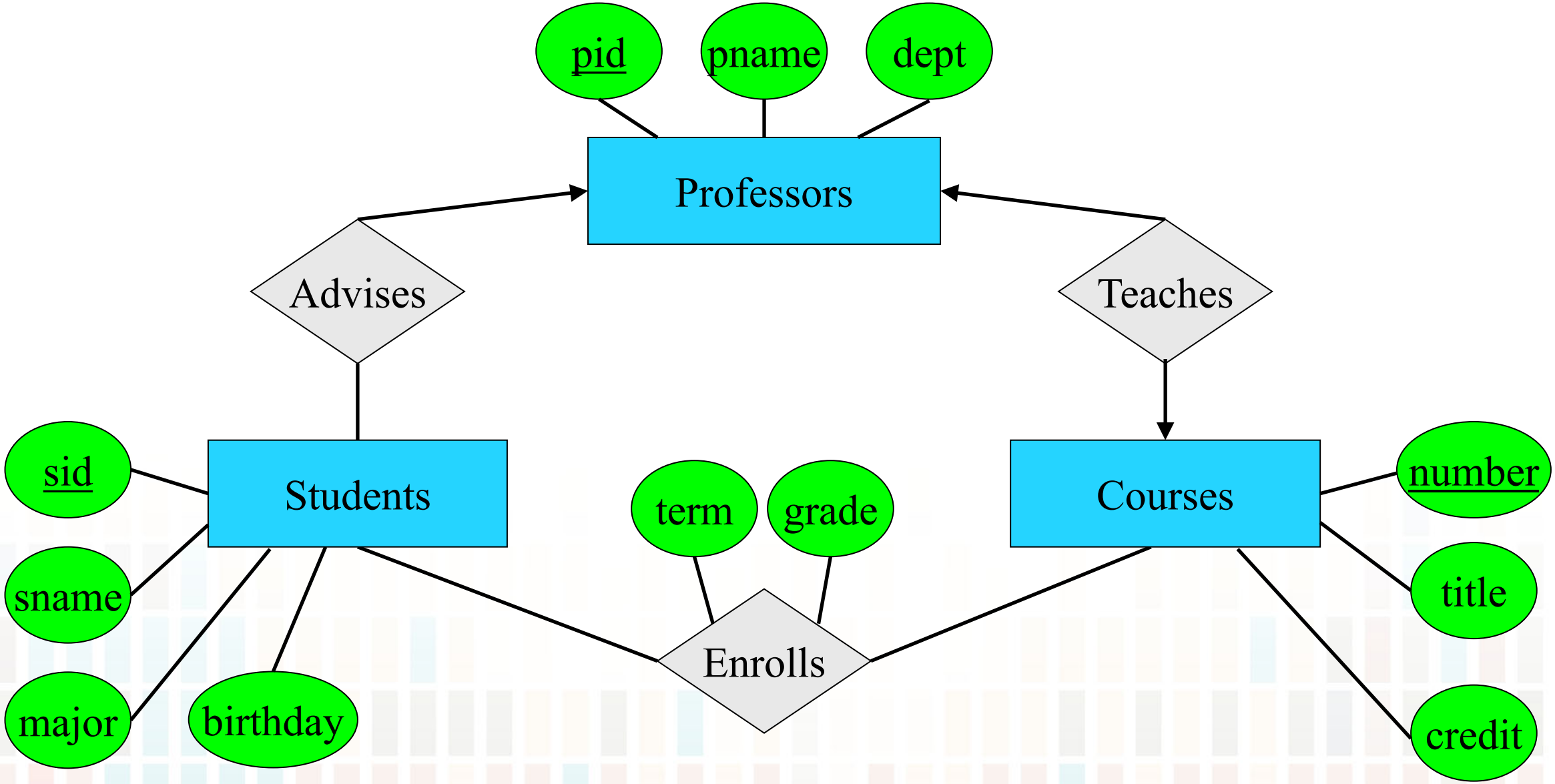

Drinker
name
addr
hobby
bar
beer
Bar
name
addr
owner
beer
Beer
name
brewer
alcohol
Beer
name
brewer
alcohol
Drinker
name
addr
hobby
bar
beer
Bar
name
addr
owner
beer
Beer
name
brewer
alcohol

σ

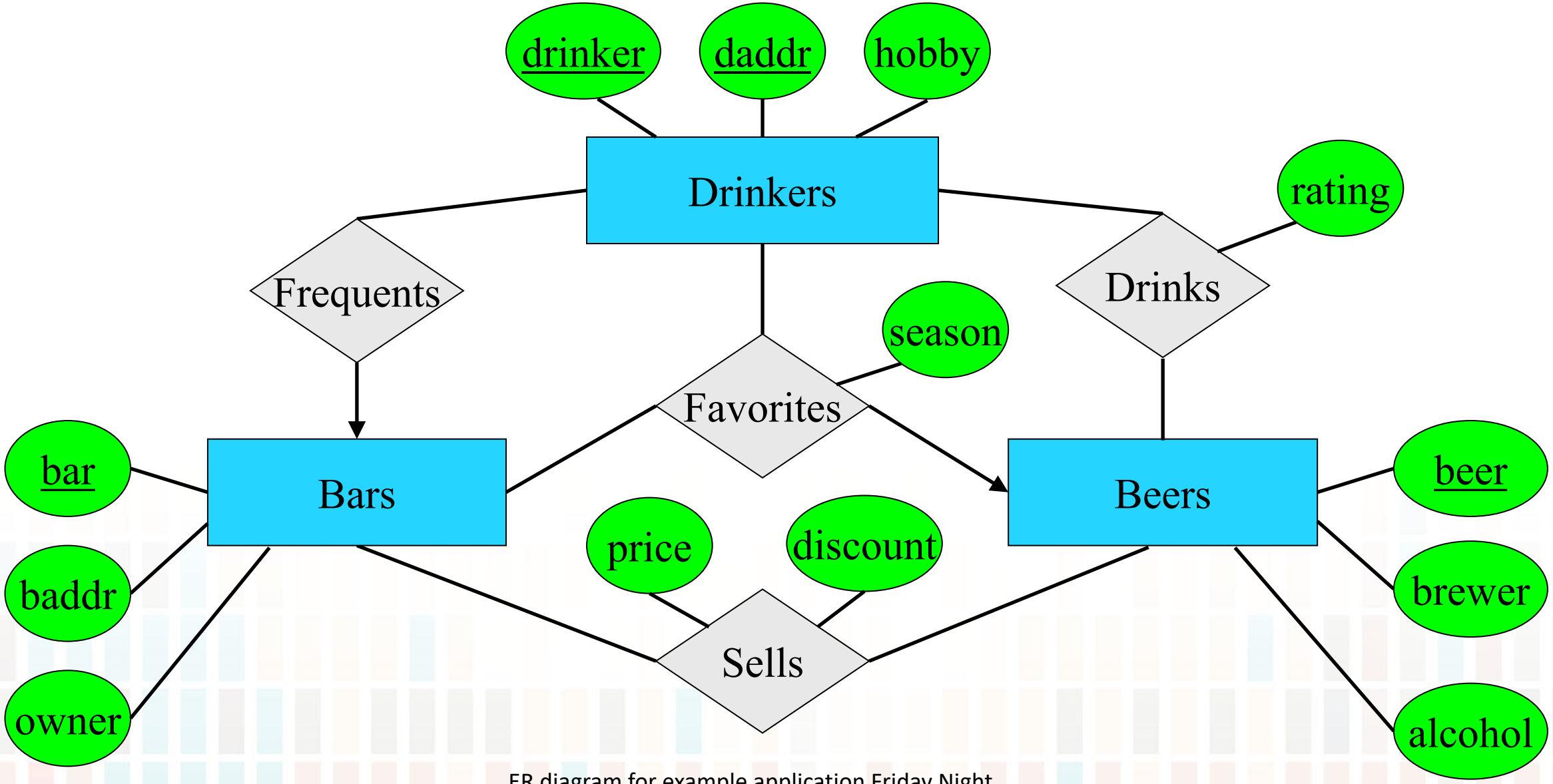


$\sigma, \pi, \bowtie, \ldots$

Academic World



Friday Night



ER diagram for example application Friday Night

Figure

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

name	color
Goose IPA	Golden
Summer Ale	Dark

name	brewer	alcohol
Sam Adams	Boston Beer	4.9

name	brewer	alcohol	color
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

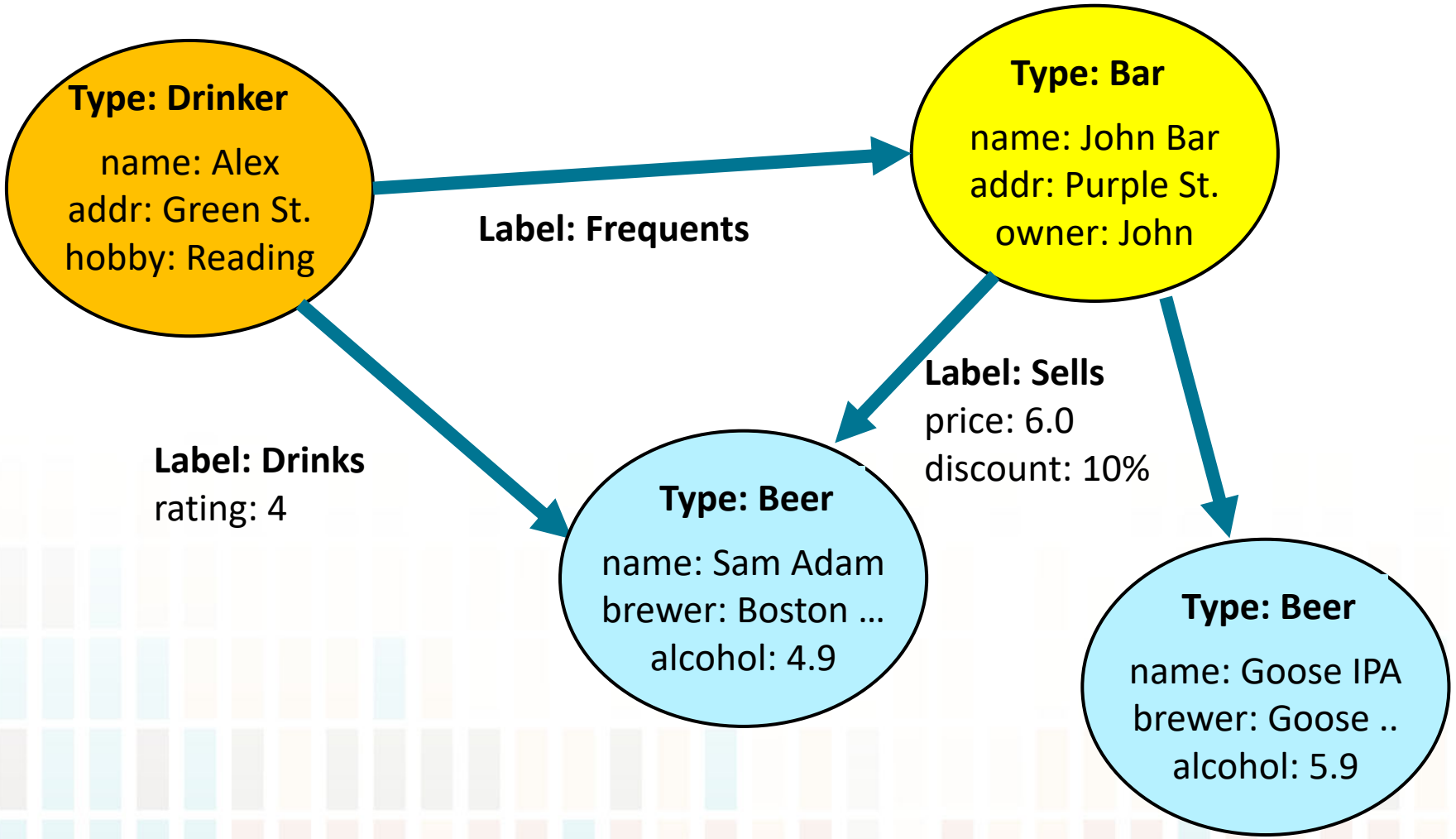
name	brewer	alcohol	aleColor
Sam Adams	Boston Beer	4.9	Null
Goose IPA	Goose Island	5.9	Golden
Summer Ale	Boston Beer	5.3	Dark

Figure

Key	Value
beer:001:name	“Sam Adams”
beer:001:brewer	”Boston Beer”
beer:001:alcohol	4.9
beer:002:name	“Goose IPA”
…	…

Key	Value
beer:001	{name: “Sam Adams”, brewer:”Boston Beer”, alcohol:4.9}
beer:002	{name:”Goose IPA”, brewer:”Goose Island”, alcohol:5.9}
…	…

Figure


Type: Drinker
name: Alex
addr: Green St.
hobby: Reading
Label: Frequents
Type: Bar
name: John Bar
addr: Purple St.
owner: John
Label: Sells
price: 6.0
discount: 10%
Label: Drinks
rating: 4
Type: Beer
name: Sam Adam
brewer: Boston …
alcohol: 4.9
Type: Beer
name: Goose IPA
brewer: Goose ..
alcohol: 5.9

Figure

Copy and paste the following to online formatter http://jsoneditoronline.org/

{ "_id": "<ObjectId1>", "name": "Samuel Adams", "brewer": { "name": "Boston Beer Company", "location": "Boston, Massachusetts" }, "alcohol": 4.9, "type": "larger", "year introduced": 1984, "variants": ["<ObjectId2>", "<ObjectId3>"]}

{ "_id": "<ObjectId2>", "name": "Samuel Adams Light", "brewer": { "name": "Boston Beer Company", "location": "Boston, Massachusetts" }, "alcohol": 3.2, "type": "larger", "year introduced": 1993}

Figure

Copy and paste the following to online formatter
http://jsoneditoronline.org/

```
{    "type" : "object",   "properties" : {            "name" :
{"type" : "string" },            "brewer" : {
"type" : "object",                  "properties" : {}            },
"alcohol" : {                  "type" : "number",
"minimum": 0,                   "maximum": 100            },
"type" : {"type" : "string"},          "year introduced" : {"type"
: "number"},          "variants" : {                  "type" :
"array",                  "items": {
"type" : "objectId",                        "minItems": 1,
"uniqueItems": true                    }          }   },
"required": ["name", "brewer", "alcohol"]}
```