W4111 – Introduction to Databases
Section 002, Fall 2023

Lecture 6: ER, Relational, SQL (V)

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Contents

Contents

- Some examples:
  - Loading IMDB datasets.
  - A little fun with indexes.
  - An example of performance.
- Codd's Rules
  - Overview
  - NULL examples
  - Metadata
- ER design patterns and advanced concepts.
- Applications: REST and web applications.

Codd's Rules

Codd's 12 Rules

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Codd's 12 Rules

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Codd's 12 Rules

Rule 3: Systematic treatment of null values:

- "Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing **missing information** and **inapplicable** information in a systematic way, independent of data type."

- Sometimes programmers and database designers are tempted to use "special values" to indicate unknow, missing or inapplicable values.
  - String: "", "NA", "UNKNOWN", …
  - Numbers: -1, 0, -9999

- Indicators can cause confusion because you have to carefully code some SQL statements to the specific, varying choices programmers made.

NULL and Correct Answers

```
In [4]: 1 %sql describe aaaaS21Examples.null_examples;
```

```
 * mysql+pymysql://dbuser:***@localhost
3 rows affected.
```

Out[4]:

Field	Type	Null	Key	Default	Extra
name	varchar(32)	NO	PRI	None	
weight	int	YES		None	
net_worth	int	YES		None	

```
In [5]: 1 %sql select * from aaaaS21Examples.null_examples;
```

```
 * mysql+pymysql://dbuser:***@localhost
4 rows affected.
```

Out[5]:

name	weight	net_worth
Joe	100	100
Larry	0	0
Pete	None	None
Tim	200	200

Without NULL, to get a correct answer:

- I must understand the domain to determine “unknown” values or know what choice a developer made.
- Explicitly include “where weight != 0” in all statements.
- And this varies from column to column, table to table, schema to schema, etc.

```
In [7]: 1 %%sql select avg(weight) as avg_weight, avg(net_worth) as avg_net_worth
        2     from aaaaS21Examples.null_examples where name in ('Joe', 'Larry', 'Tim')
```

```
 * mysql+pymysql://dbuser:***@localhost
1 rows affected.
```

Out[7]:

avg_weight	avg_net_worth
100.0000	100.0000

```
In [9]: 1 %%sql select avg(weight) as avg_weight, avg(net_worth) as avg_net_worth
        2     from aaaaS21Examples.null_examples where name in ('Joe', 'Pete', 'Tim')
```

```
 * mysql+pymysql://dbuser:***@localhost
1 rows affected.
```

Out[9]:

avg_weight	avg_net_worth
150.0000	150.0000

Metadata and Catalog

- ‘Metadata is "data that provides information about other data". In other words, it is "data about data". Many distinct types of metadata exist, including descriptive metadata, structural metadata, administrative metadata, reference metadata and statistical metadata.’ (https://en.wikipedia.org/wiki/Metadata)

- “The database catalog of a database instance consists of metadata in which definitions of database objects such as base tables, views (virtual tables), synonyms, value ranges, indexes, users, and user groups are stored. ... ...

  The SQL standard specifies a uniform means to access the catalog, called the INFORMATION_SCHEMA, but not all databases follow this ...” (https://en.wikipedia.org/wiki/Database_catalog)

- Codd’s Rule 4: Dynamic online catalog based on the relational model:
  - The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example: **create table** *instructor* (
  ID **char**(5),
  name **varchar**(20)**,**
  dept_name **varchar**(20),
  salary **numeric**(8,2))

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

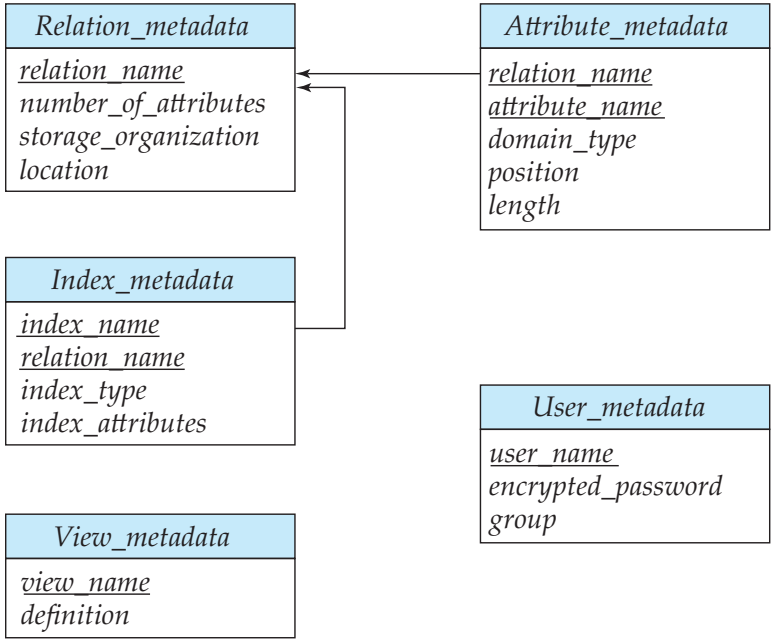
Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as

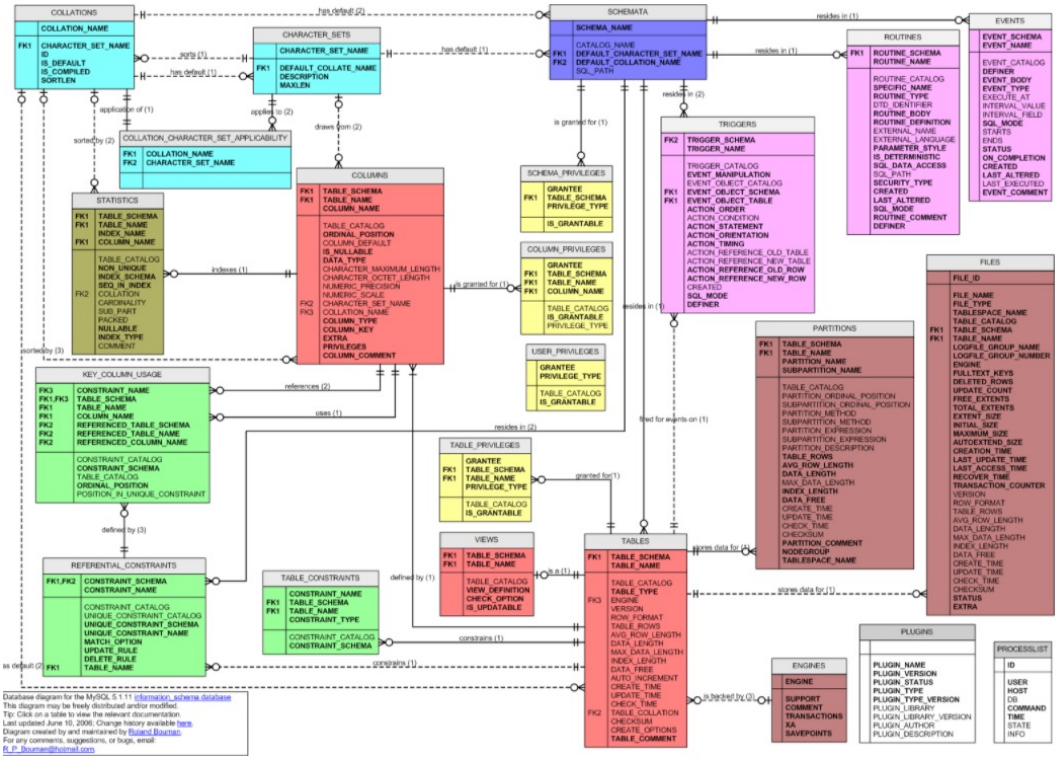
- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices (Chapter 14)

Relational Representation of System Metadata

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory



MySQL Catalog (Information_Schema)



Some of the MySQL Information Schema Tables:

- 'ADMINISTRABLE_ROLE_AUTHORIZATIONS'
- 'APPLICABLE_ROLES'
- 'CHARACTER_SETS'
- 'CHECK_CONSTRAINTS'
- 'COLUMN_PRIVILEGES'
- 'COLUMN_STATISTICS'
- 'COLUMNS'
- 'ENABLED_ROLES'
- 'ENGINES'
- 'EVENTS'
- 'FILES'
- 'KEY_COLUMN_USAGE'
- 'PARAMETERS'
- 'REFERENTIAL_CONSTRAINTS'
- 'RESOURCE_GROUPS'
- 'ROLE_COLUMN_GRANTS'
- 'ROLE_ROUTINE_GRANTS'
- 'ROLE_TABLE_GRANTS'
- 'ROUTINES'
- 'SCHEMA_PRIVILEGES'
- 'STATISTICS'
- 'TABLE_CONSTRAINTS'
- 'TABLE_PRIVILEGES'
- 'TABLES'
- 'TABLESPACES'
- 'TRIGGERS'
- 'USER_PRIVILEGES'
- 'VIEW_ROUTINE_USAGE'
- 'VIEW_TABLE_USAGE'
- 'VIEWS'

- CREATE and ALTER statements modify the data.
- DBMS reads information:
  - Parsing
  - Optimizer
  - etc.

ER Modeling
Design Patterns and Advanced Concepts

Model and Concepts

Weak Entity Sets

- Consider a *section* entity, which is uniquely identified by a *course_id*, *semester, year*, and *sec_id*.
- Clearly, section entities are related to course entities. Suppose we create a relationship set *sec_course* between entity sets *section* and *course*.
- Note that the information in *sec_course* is redundant, since *section* already has an attribute *course_id*, which identifies the course with which the section is related.
- One option to deal with this redundancy is to get rid of the relationship s*ec_course*; however, by doing so the relationship between *section* and *course* becomes implicit in an attribute, which is not desirable.

Weak Entity Sets (Cont.)

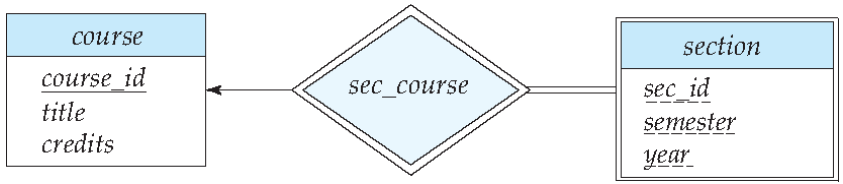
- An alternative way to deal with this redundancy is to not store the attribute *course_id* in the *section* entity and to only store the remaining attributes *section_id*, *year*, and *semester.*
  - However, the entity set *section* then does not have enough attributes to identify a particular *section* entity uniquely
- To deal with this problem, we treat the relationship *sec_course* as a special relationship that provides extra information, in this case, the *course_id*, required to identify *section* entities uniquely.
- A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity.

Weak Entity Sets (Cont.)

- An entity set that is not a weak entity set is termed a **strong entity set**.
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set.
- The identifying entity set is said to **own** the weak entity set that it identifies.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.
- Note that the relational schema we eventually create from the entity set *section* does have the attribute *course_id*, for reasons that will become clear later, even though we have dropped the attribute *course_id* from the entity set *section.*

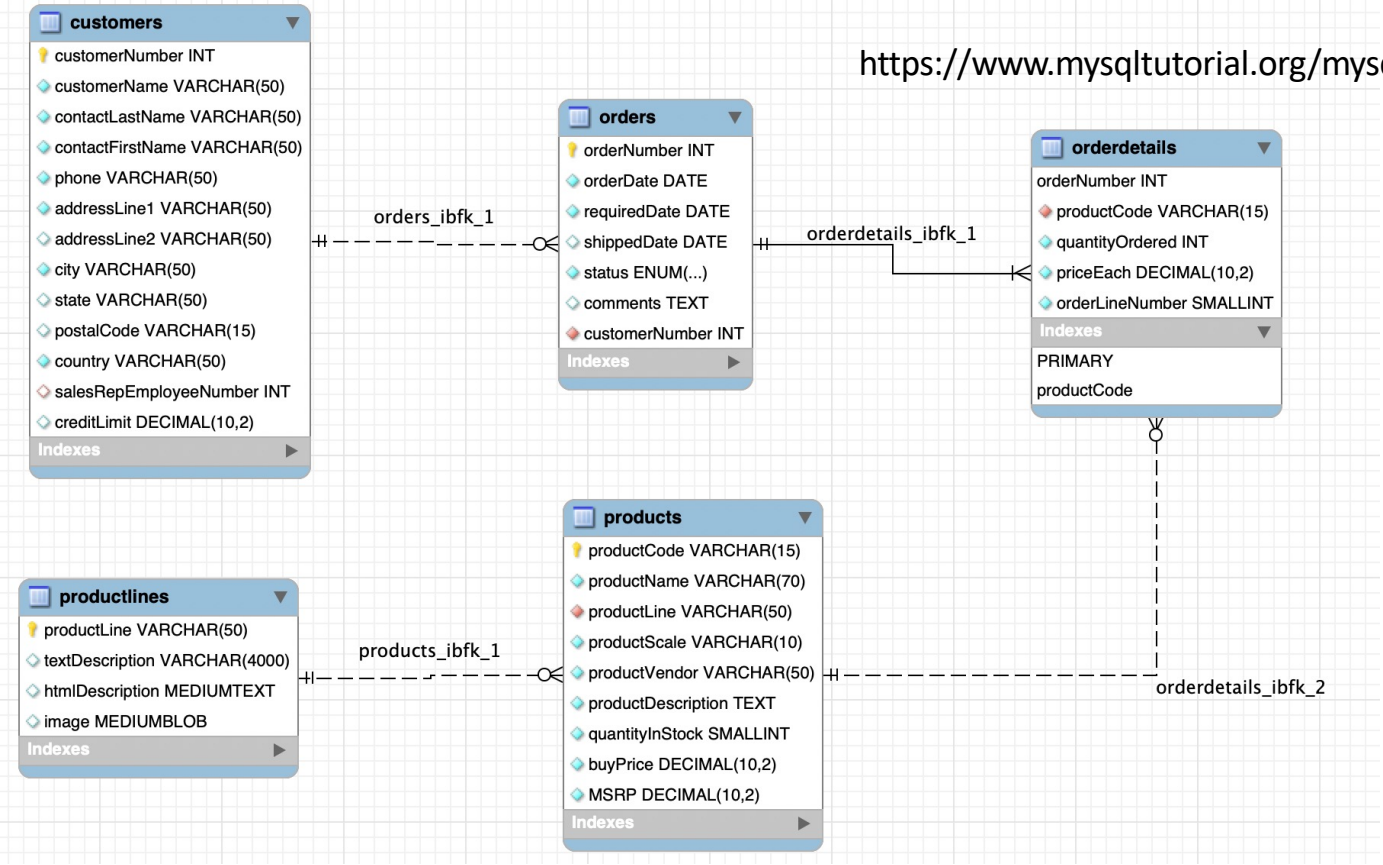
Expressing Weak Entity Sets

- In E-R diagrams, a weak entity set is depicted via a double rectangle.
- We underline the discriminator of a weak entity set with a dashed line.
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.
- Primary key for *section* – (*course_id, sec_id, semester, year*)



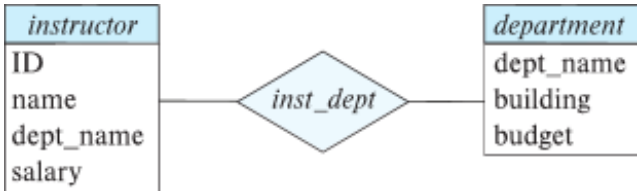
An Example – Classic Models

https://www.mysqltutorial.org/mysql-sample-database.aspx/



Redundant Attributes

- Suppose we have entity sets:
  - *instructor*, with attributes: *ID*, *name*, *dept_name, salary*
  - *department,* with attributes: *dept_name, building, budget*
- We model the fact that each instructor has an associated department using a relationship set *inst_dept*
- The attribute *dept_name* in *instructor* replicates information present in the relationship and is therefore redundant
  - and needs to be removed.
- BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see later.



Design Alternatives

- In designing a database schema, we must ensure that we avoid two major pitfalls:
  - Redundancy: a bad design may result in repeat information.
    - **Redundant representation of information may lead to data inconsistency among the various copies of information**
  - Incompleteness: a bad design may make certain aspects of the enterprise difficult or impossible to model.
- Avoiding bad designs is not enough. There may be a large number of good designs from which we must choose.

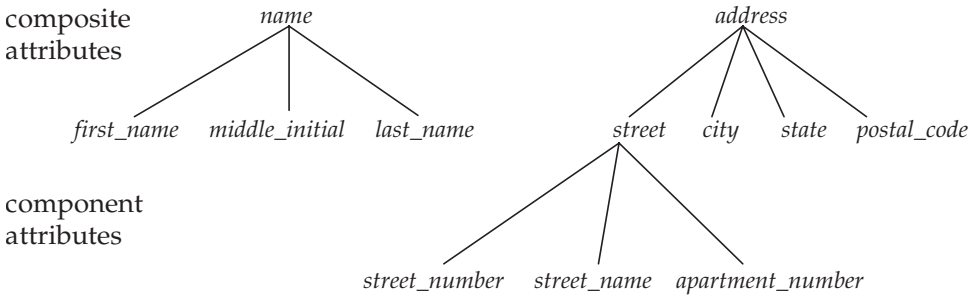
Emphasis Added

Complex Attributes

- Attribute types:
  - **Simple** and **composite** attributes.
  - **Single-valued** and **multivalued** attributes
    - Example: multivalued attribute: *phone_numbers*
  - **Derived** attributes
    - Can be computed from other attributes
    - Example: age, given date_of_birth
- **Domain** – the set of permitted values for each attribute

Composite Attributes

- Composite attributes allow us to divided attributes into subparts (other attributes).



Example from IMDB

- Consider name_basics

10 rows | Tx: Auto | DDL | CSV

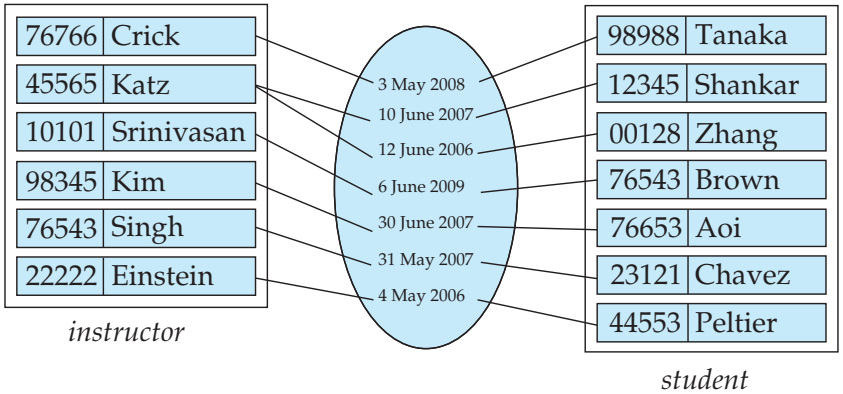
	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
1	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0050419,tt0031983,tt0072308,tt0053137
2	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0071877,tt0117057,tt0037382,tt0038355
3	nm0000003	Brigitte Bardot	1934	<null>	actress,soundtrack,music_department	tt0049189,tt0056404,tt0057345,tt0054452
4	nm0000004	John Belushi	1949	1982	actor,soundtrack,writer	tt0077975,tt0072562,tt0080455,tt0078723
5	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0050986,tt0060827,tt0069467,tt0050976
6	nm0000006	Ingrid Bergman	1915	1982	actress,soundtrack,producer	tt0077711,tt0038109,tt0034583,tt0036855
7	nm0000007	Humphrey Bogart	1899	1957	actor,soundtrack,producer	tt0043265,tt0034583,tt0042593,tt0037382
8	nm0000008	Marlon Brando	1924	2004	actor,soundtrack,director	tt0078788,tt0068646,tt0070849,tt0047296
9	nm0000009	Richard Burton	1925	1984	actor,soundtrack,producer	tt0061184,tt0087803,tt0057877,tt0059749
10	nm0000010	James Cagney	1899	1986	actor,soundtrack,director	tt0029870,tt0035575,tt0042041,tt0055256

- There
  - Is one composite attribute, primaryName.
  - Are two multivalued attributes: primaryProfession, knownForTitles
  - knownForTitles is also tricky, which we will see.
  - Names are also a little tricky

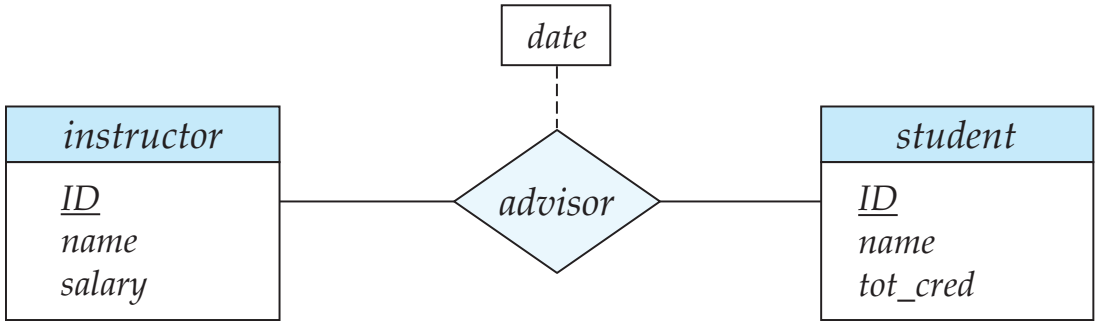
Switch to notebook

Relationship Sets (Cont.)

- An attribute can also be associated with a relationship set.
- For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor

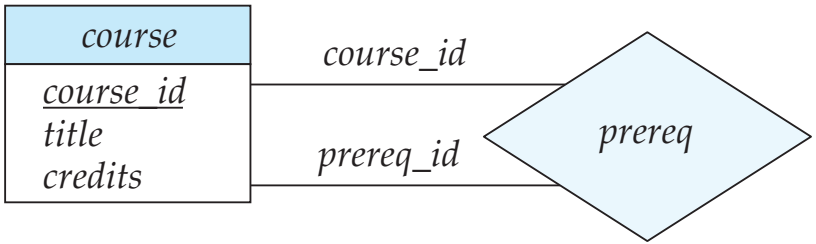


Relationship Sets with Attributes



Roles

- Entity sets of a relationship need not be distinct
  - Each occurrence of an entity set plays a “role” in the relationship
- The labels “*course_id*” and “*prereq_id*” are called **roles**.

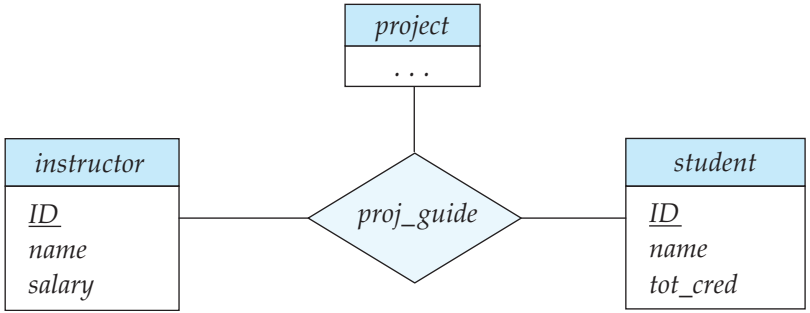


Degree of a Relationship Set

- Binary relationship
  - involve two entity sets (or degree two).
  - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
  - Example: *students* work on research *projects* under the guidance of an *instructor*.
  - relationship *proj_guide* is a ternary relationship between *instructor, student,* and *project*

Non-binary Relationship Sets

- Most relationship sets are binary
- There are occasions when it is more convenient to represent relationships as non-binary.
- E-R Diagram with a Ternary Relationship

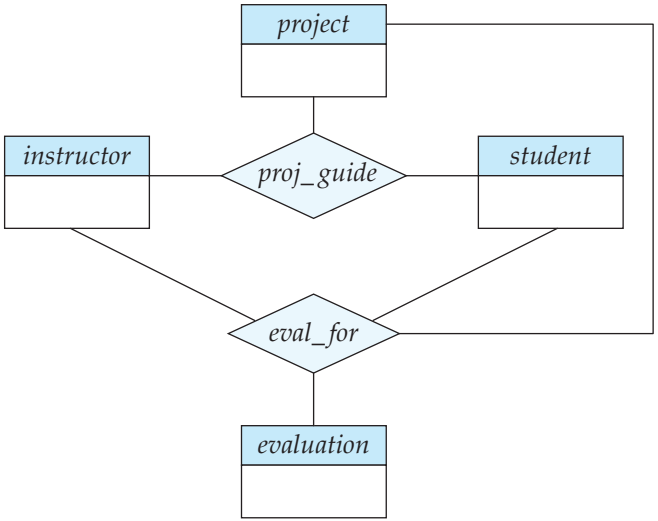


Mapping Cardinality Constraints

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Aggregation

- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project

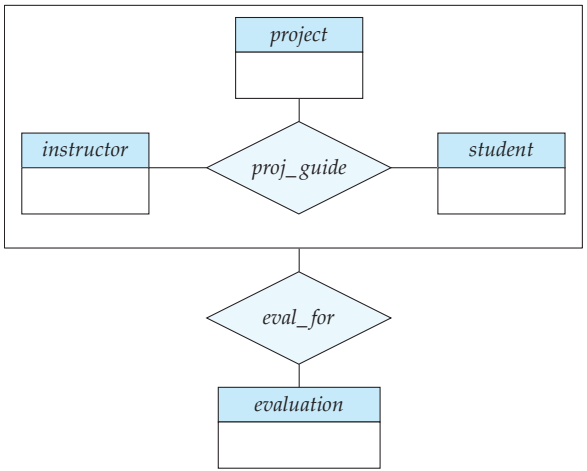


Aggregation (Cont.)

- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

Aggregation (Cont.)

- Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, instructor, project combination may have an associated evaluation



Reduction to Relational Schemas

- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship,
  - The primary key of the associated entity set
  - Any descriptive attributes
- In our example:
  - The schema *eval_for* is:

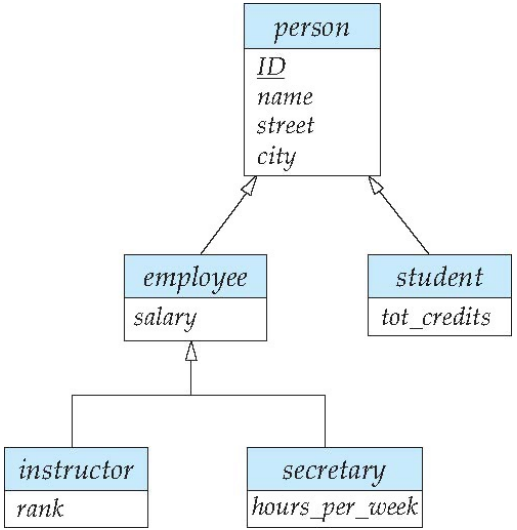
    eval_for (*s_ID, project_id, i_ID, evaluation_id*)
  - The schema *proj_guide* is redundant.

Specialization

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (e.g., *instructor* "is a" *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

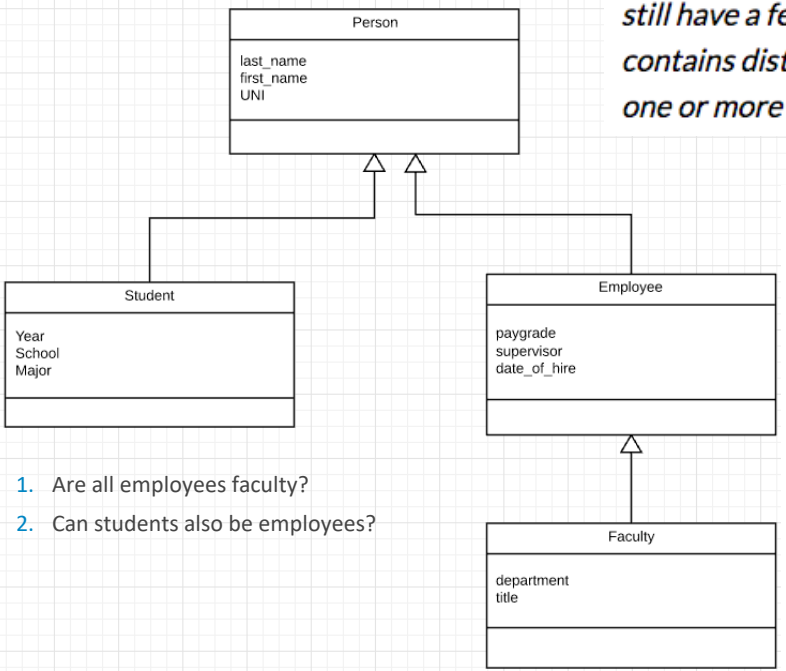
Specialization Example

- **Overlapping** – *employee* and *student*
- **Disjoint** – *instructor* and *secretary*
- Total and partial



Inheritance/Specialization

In the process of designing our entity relationship diagram for a database, we may find that attributes of two or more entities overlap, meaning that these entities seem very similar but still have a few differences. In this case, we may create a subtype of the parent entity that contains distinct attributes. A parent entity becomes a supertype that has a relationship with one or more subtypes.



1. Are all employees faculty?
2. Can students also be employees?

The subclass association line is labeled with specialization constraints. Constraints are described along two dimensions:

1 incomplete/complete

- In an **incomplete** specialization only some instances of the parent class are specialized (have unique attributes). Other instances of the parent class have only the common attributes.
- In a **complete** specialization, every instance of the parent class has one or more unique attributes that are not common to the parent class.

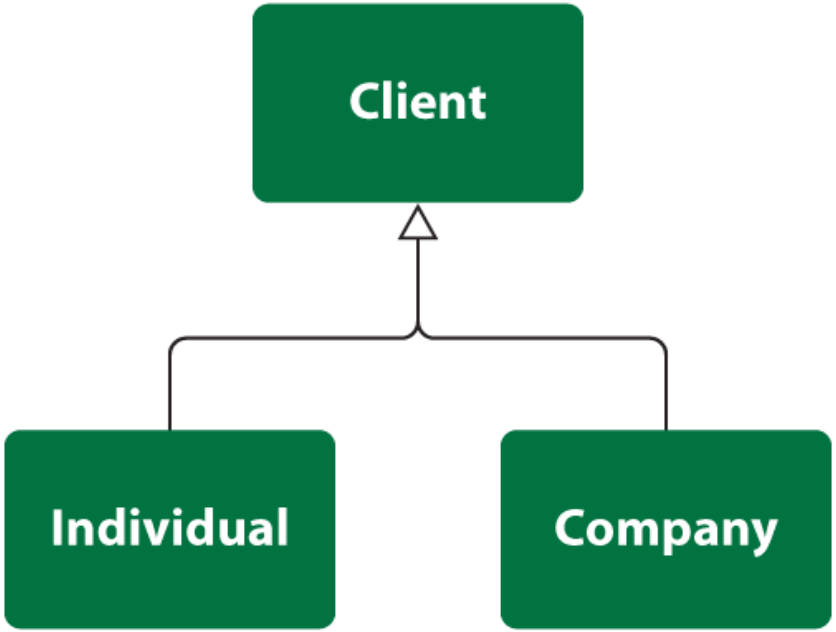
2 disjoint/overlapping

- In a **disjoint** specialization, an object could be a member of only one specialized subclass.
- In an **overlapping** specialization, an object could be a member of more than one specialized subclass.

http://www.vertabelo.com/blog/technical-articles/inheritance-in-a-relational-database

Specialization

In class Client we distinguish two subtypes: Individual and Company. This specialization is disjoint (client can be an individual or a company) and complete (these are all possible subtypes for supertype).

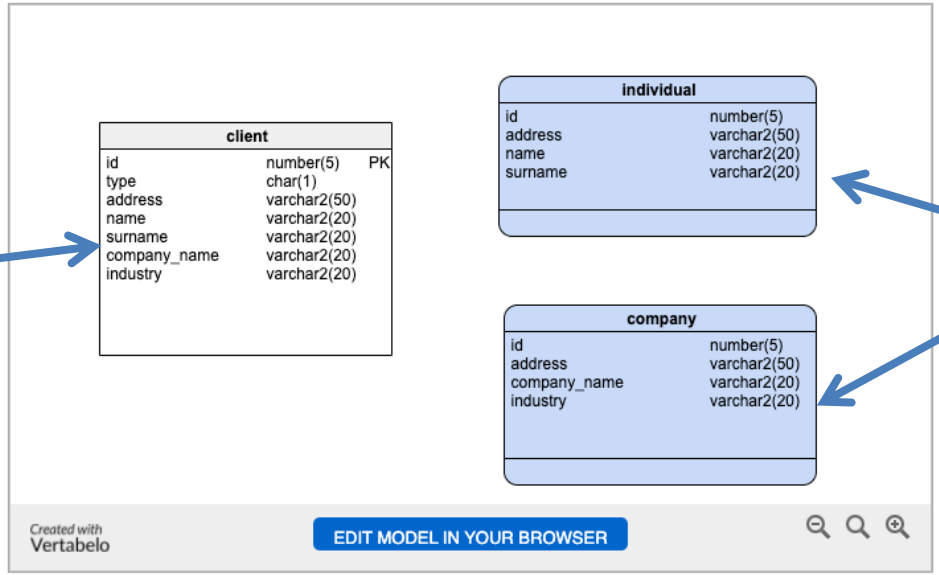


One Table

One table implementation

In a one table implementation, table `client` has attributes of both types.

The diagram below shows the table `client` and two views: `individual` and `company`:



Tables

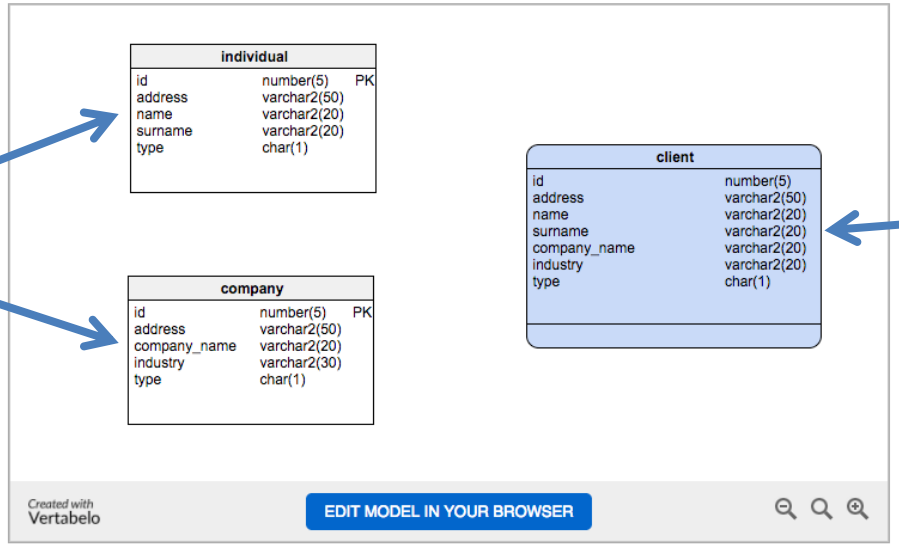
Views

Two Table

Two-table implementation

In a two-table implementation, we create a table for each of the subtypes. Each table gets a column for all attributes of the supertype and also a column for each attribute belonging to the subtype. Access to information in this situation is limited, that's why it is important to create a view that is the union of the tables. We can add an additional attribute called 'type' that describes the subtype.

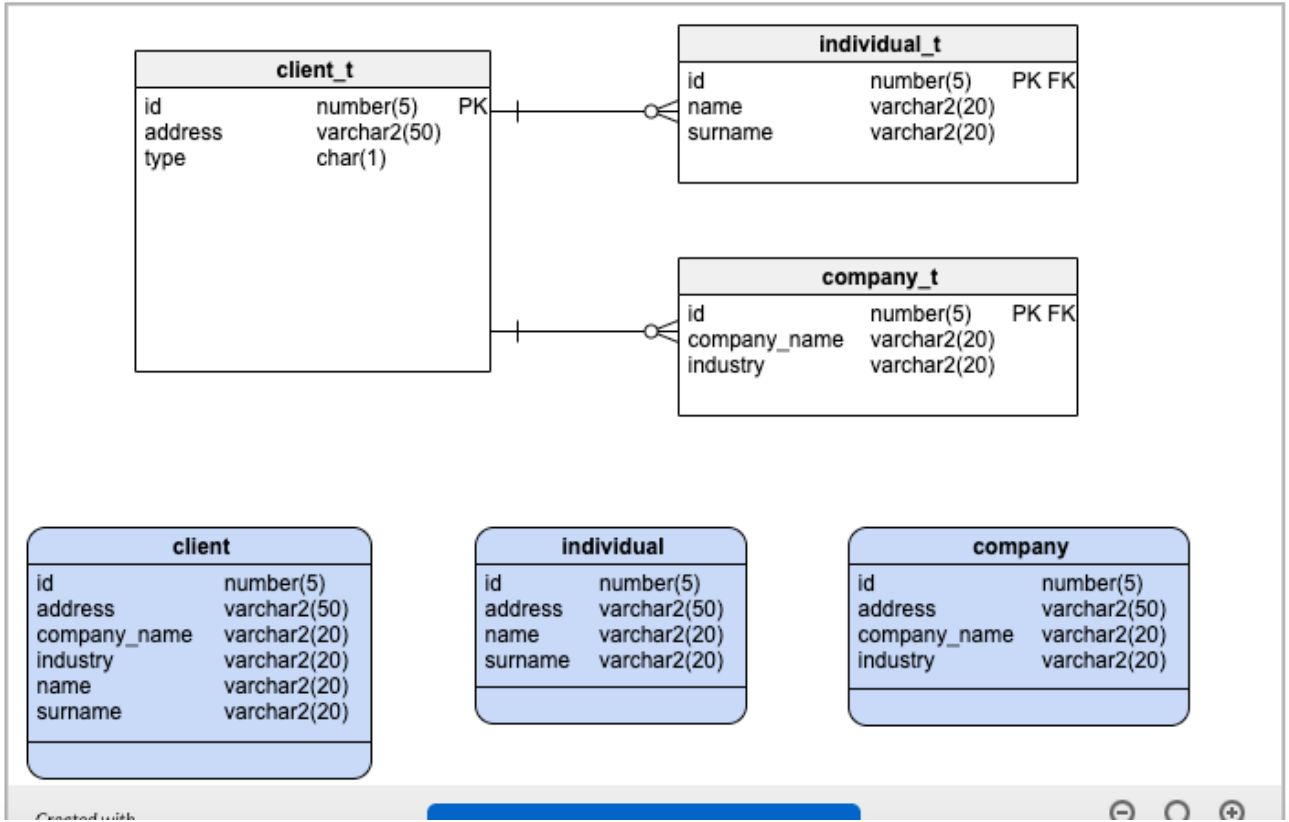
The diagram below presents two tables, `individual` and `company`, and a view (the blue one) called `client`.



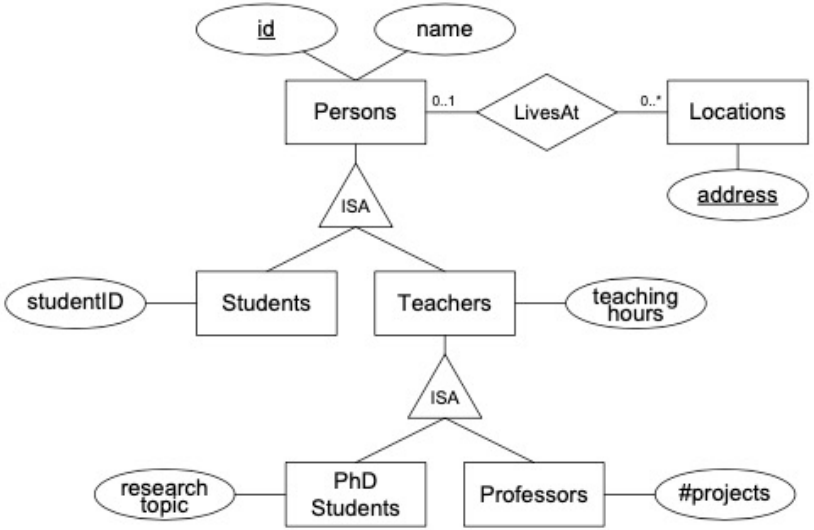
Tables

View

Three Table

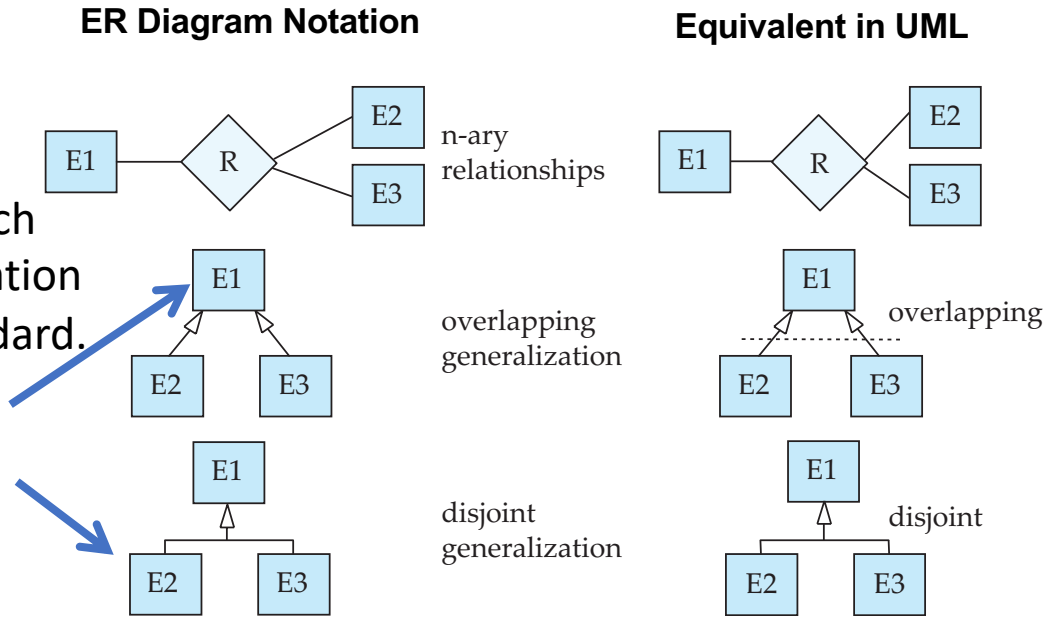


ISA Relationship



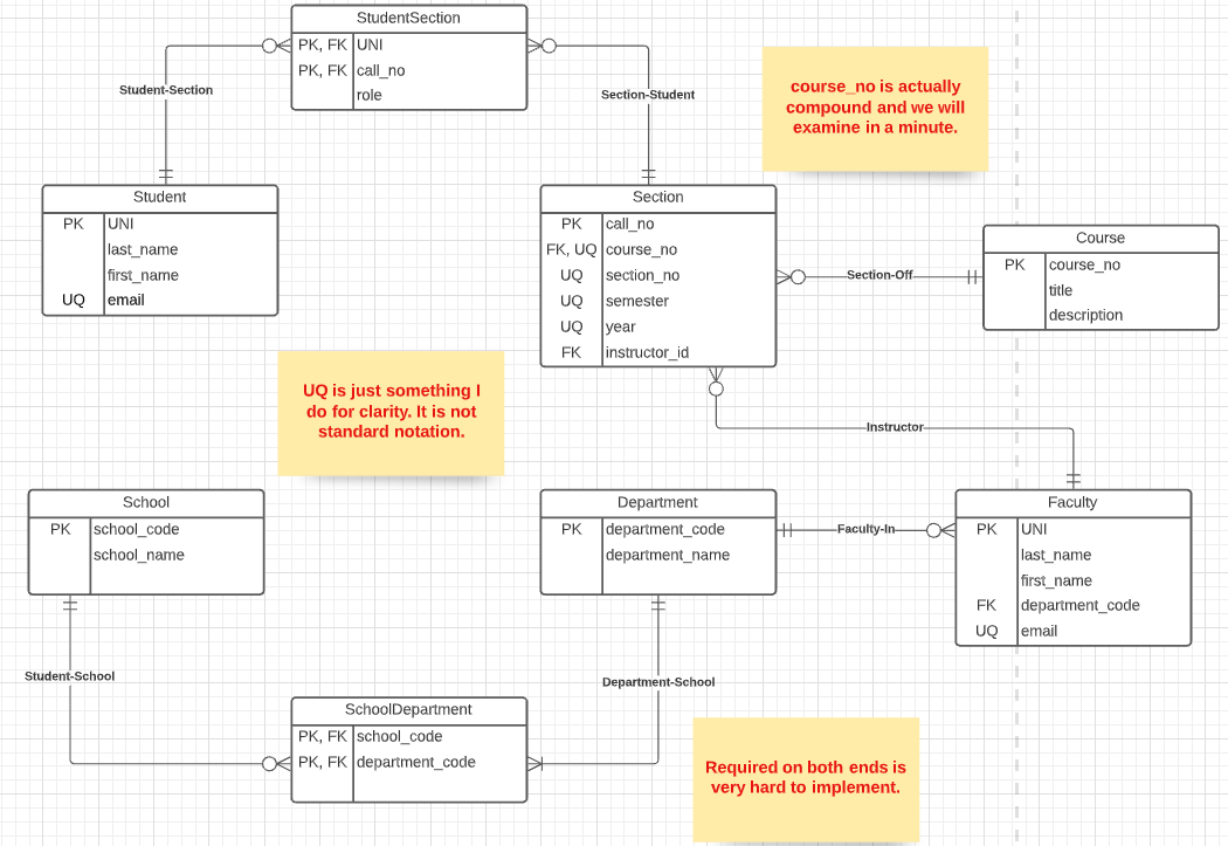
ER vs. UML Class Diagrams

I use this approach in Crow's Foot Notation but that is not standard.



* Generalization can use merged or separate arrows independent of disjoint/overlapping

Approximate Model



Course Number

https://www.cc-seas.columbia.edu/sites/dsa/files/handbooks/Columbia%20Key%20to%20Course%20Listing.pdf

A	*Architecture, Planning, and Preservation**
B	*Business**
BC	**Barnard College**
C	**Columbia College**
D	*Dentistry***
E	**Engineering and Applied Science**
F	**General Studies**
G	**Graduate School of Arts and Sciences**
H	*Reid Hall, Paris***
I	*Berlin Consortium Program***
J	*Journalism**
K	*Continuing Education***
L	*Law***
M	*Medicine***
N	*Nursing***
O	*Union Theological***
P	*School of Public Health**
R	*School of the Arts**
S	**Summer Session**
T	*Social Work**
TA–TZ	*Teachers College**
U	*International and Public Affairs**
V	**Interschool course with Barnard**
W	**Interfaculty course**
X	**Barnard College**
Z	*American Language Program(no credit)***

Key to Columbia Course Listings

(Example: ECON W1105 001 Principles of Economics, 4 pts)

	Example	Description
Call # (5 digit number)	***16238***	This 5 digit code is assigned to individual courses and is specific to each semester.
Department Code (4 letter code)	***ECON***	This 4 letter code represents the Academic Department that manages the course.
Course Number (1 capital letter followed by 4 digit code)	***W***	The *capital letters* indicate the instructor teaching the course and their affiliation with a **division, school** or **affiliate** of the **University**. Unless otherwise noted, courses numbers beginning with the following letters are generally open to CC/SEAS undergraduate students: BC Barnard College C Columbia College E Engineering and Applied Science F General Studies G Graduate School of Arts and Sciences V Interschool course with Barnard W Interfaculty course X Barnard College
	1105	The *first digit* indicates the **level of the course**. Generally, levels are indicated as: 0 Course that cannot be credited toward any degree 1 Undergraduate course, introductory 2 Undergraduate course, intermediate 3 Undergraduate course, advanced 4 Graduate course that is open to qualified undergraduates 6 Graduate course 8 Graduate course, advanced 9 Graduate research course or seminar
Course Section	***001***	Based on course demand, some academic departments offer the same course during 2 or more different time slots. Each time slot is assigned a different section number.
Points/Credits	***4***	The term "points" and "credits" are often used interchangeably and is generally related to the number of classroom contact hours.

Columbia Open Data Service

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

OPEN DATA SERVICE

News and Updated About Data Feeds Request a New Feed

Data Feed Service

Columbia University offers data feeds in programming-friendly formats for research and academic purposes. The Open Data Feed Service currently offers the following data feeds:

- Course Information
- Athletic Schedule
- Academic Commons
- CLIO - Library Catalog Data
- Textbooks Feed

Data feed information includes the feed's refresh schedule, data diagrams, table and data element documentation, data training, and data security. More details on each feed is available on each feed's page.

Associative Entity

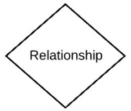
- The ER model represents “associations/relationships” as
  - First class “things”
  - That are different from “entities.”
- The SQL model does not have “relationships” or associations as first class types. You have
  - Tables
  - Columns
  - Keys
  - Constraints
  - … …
- You can implement some “relationships” using foreign keys. Others require something more complex – an *associative entity.*

Associative entity

Associative entities relate the instances of several entity types. They also contain attributes specific to the relationship between those entity instances.

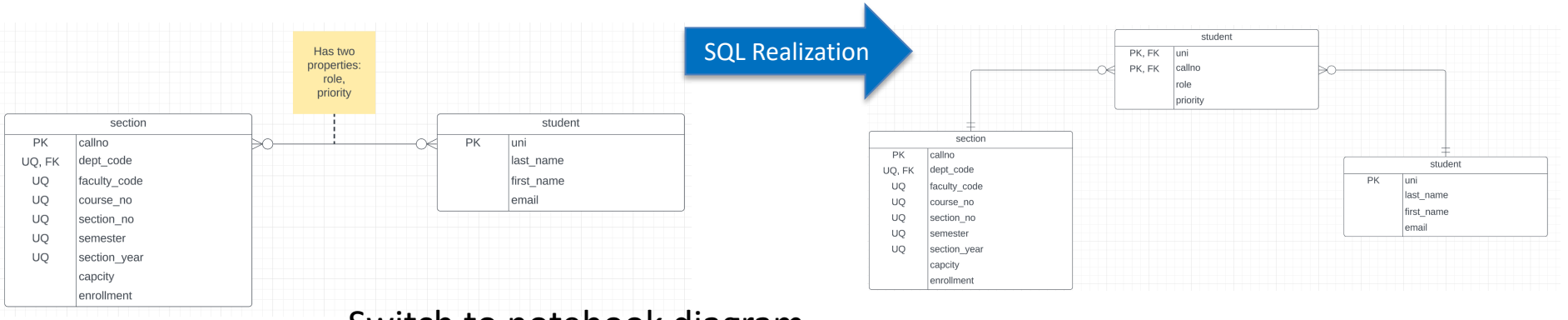
ERD relationship symbols

Within entity-relationship diagrams, relationships are used to document the interaction between two entities. Relationships are usually verbs such as assign, associate, or track and provide useful information that could not be discerned with just the entity types.

Relationship Symbol	Name	Description
	Relationship	Relationships are associations between or among entities.
	Weak relationship	Weak Relationships are connections between a weak entity and its owner.

Associative Entity

- "An associative entity is a term used in relational and entity–relationship theory. A relational database requires the implementation of a base relation (or base table) to resolve many-to-many relationships. A base relation representing this kind of entity is called, informally, an associative table." (https://en.wikipedia.org/wiki/Associative_entity)
- Consider *Students – Sections:*
  - This is many-to-many. There is not way to implement in SQL. You see this in the *Advises* table in the sample database.
  - The "relationship/association" has properties that are not attributes of the connected entities.



Switch to notebook diagram.

Do Some Examples

Do Some Examples

- Composite and Multi-Valued Attributes
  - Composite:
    - *primaryName*.
    - Show simple example of splitting with a function.
    - Show HumanName and flow.
  - Multi-Valued:
    - *knownFor*.
    - Show pattern: Create a table. Associative entity.
- Aggregation: Class-project example.
- Inheritance examples.

REST

Data Modeling Concepts and REST

Almost any data model has the same core concepts:

- Types and instances:
  - Entity Type: A definition of a type of thing with properties and relationships.
  - Entity Instance: A specific instantiation of the Entity Type
  - Entity Set Instance: An Entity Type that:
    - Has properties and relationships like any entity, but ...
    - Has at least one *special relationship* – ***contains.***
- Operations, minimally CRUD, that manipulate entity types and instances:
  - Create
  - Retrieve
  - Update
  - Delete
  - Reference/Identify/... ...

REST (https://www.tutorialspoint.com/restful/restful_introduction.htm)

What is REST architecture?

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

- **GET** – Provides a read only access to a resource.
- **POST** – Used to create a new resource.
- **DELETE** – Used to remove a resource.
- **PUT** – Used to update a existing resource or create a new resource.

REST (https://www.tutorialspoint.com/restful/restful_introduction.htm)

Introduction to RESTFul web services

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

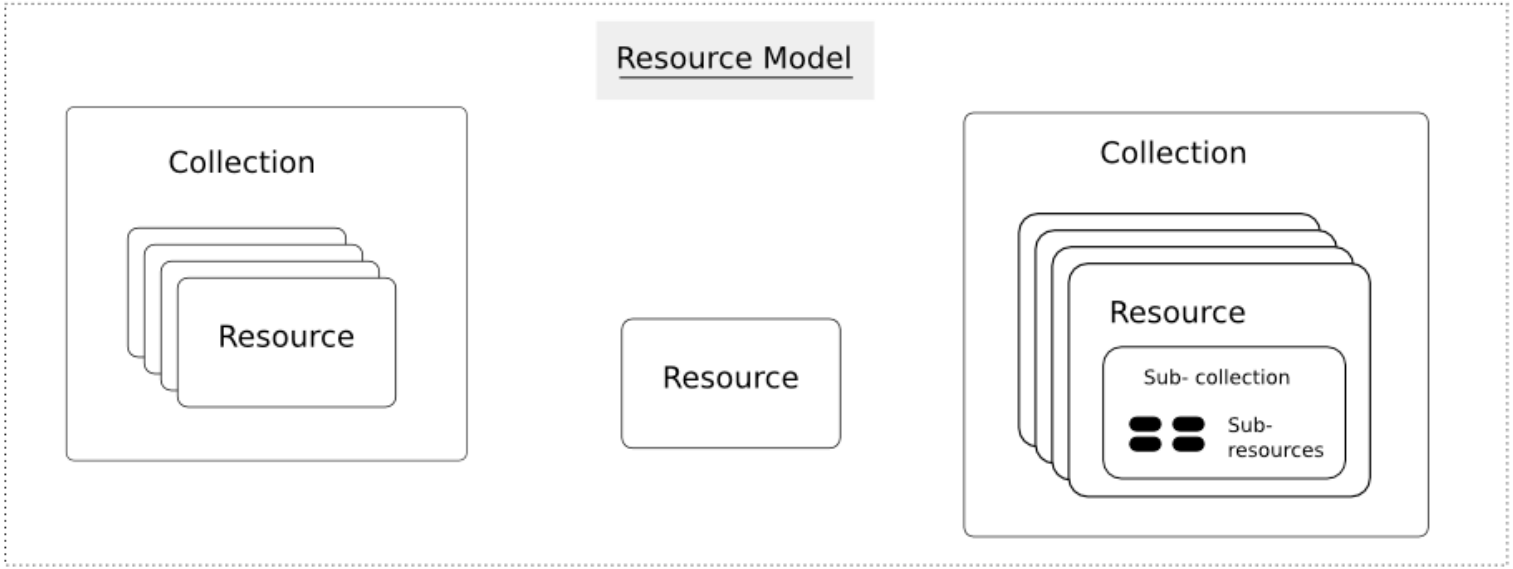
Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

Creating RESTFul Webservice

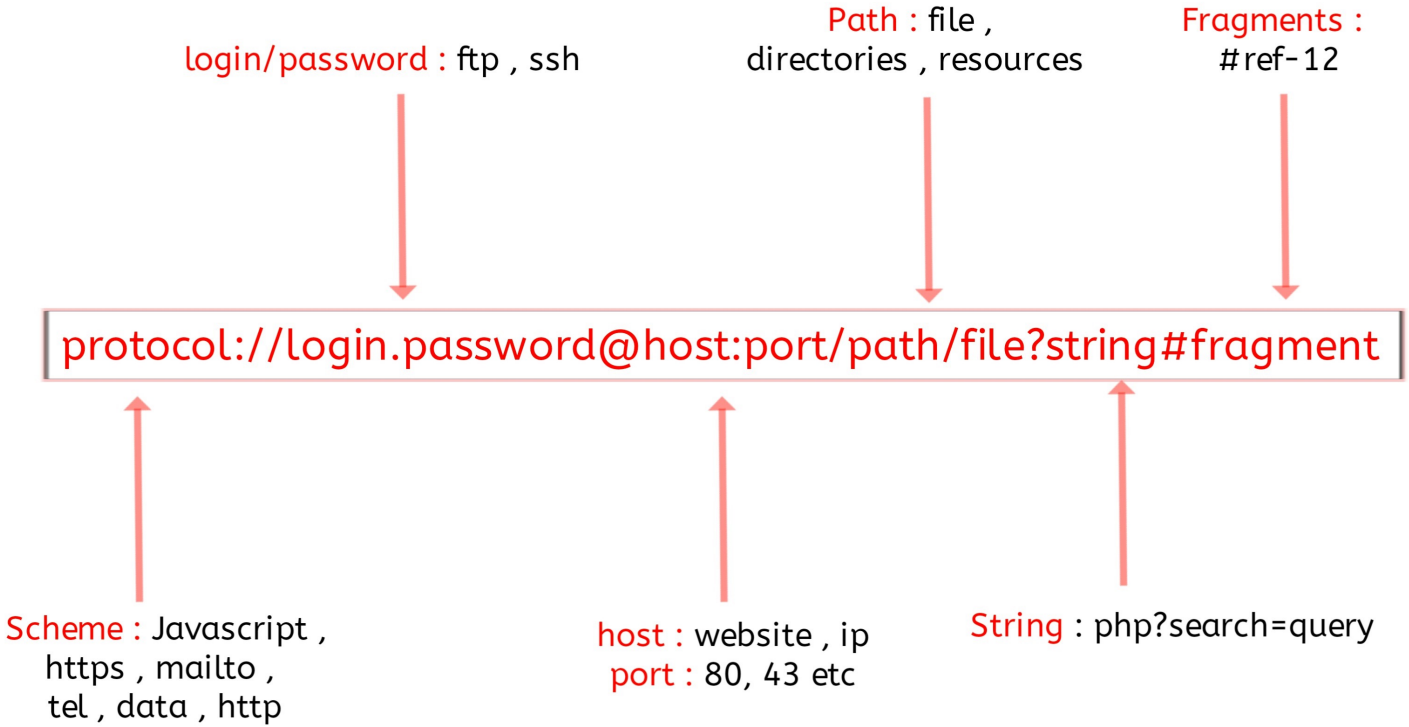
In next chapters, we'll create a webservice say user management with following functionalities –

Sr.No.	URI	HTTP Method	POST body	Result
1	/UserService/users	GET	empty	Show list of all the users.
2	/UserService/addUser	POST	JSON String	Add details of new user.
3	/UserService/getUser/:id	GET	empty	Show details of a user.

REST and Resources



URLs



jdbc:mysql://columbia-examples.ckkqqktwkcji.us-east-1.rds.amazonaws.com:3306

Simplistic, Conceptual Mapping (Examples)

REST Method	Resource Path	Relational Operation	DB Resource
DELETE	/people	DROP TABLE	people table
POST	/people	INSERT INTO PEOPLE (...) VALUES(...)	people table people row
GET	/people/21	SHOW KEYS FROM people ...; SELECT * FROM people WHERE playerID= 21	people row
GET	/people/21/batting	SELECT batting.* FROM people JOIN batting USING(playerID) WHERE playerID=21	
GET	/people/21/batting/2004_1	SELECT batting.* FROM people JOIN batting USING(playerID) WHERE playerID=21 AND yearID=2004 AND stint=1	

Application Architecture

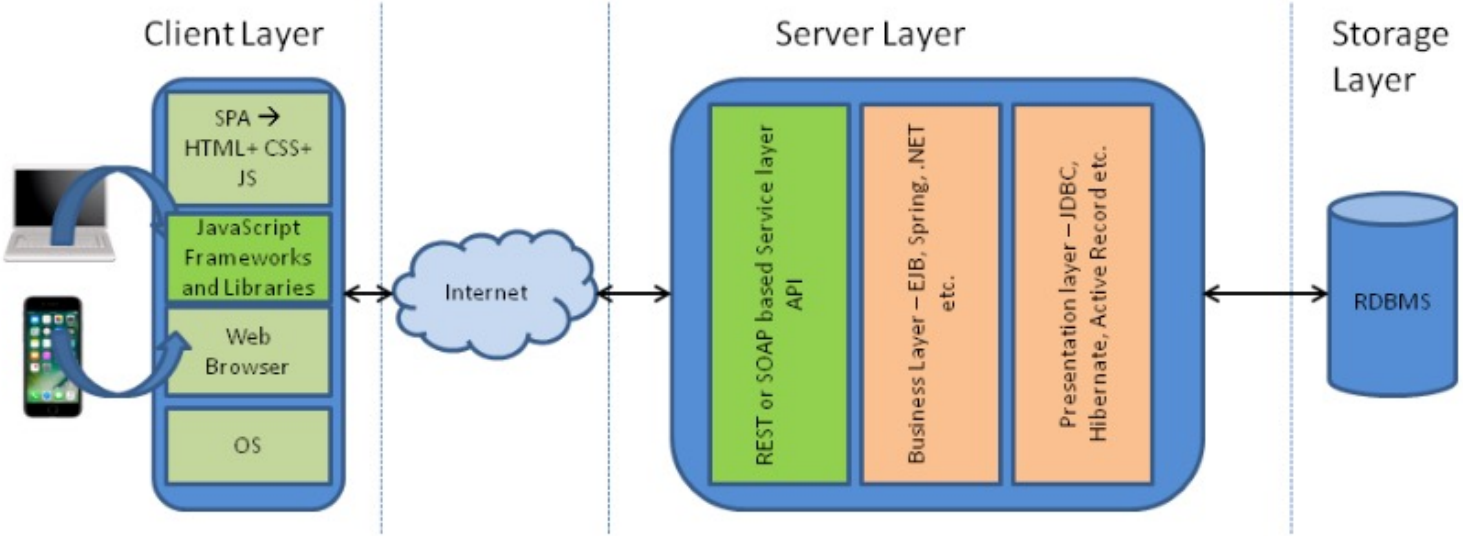


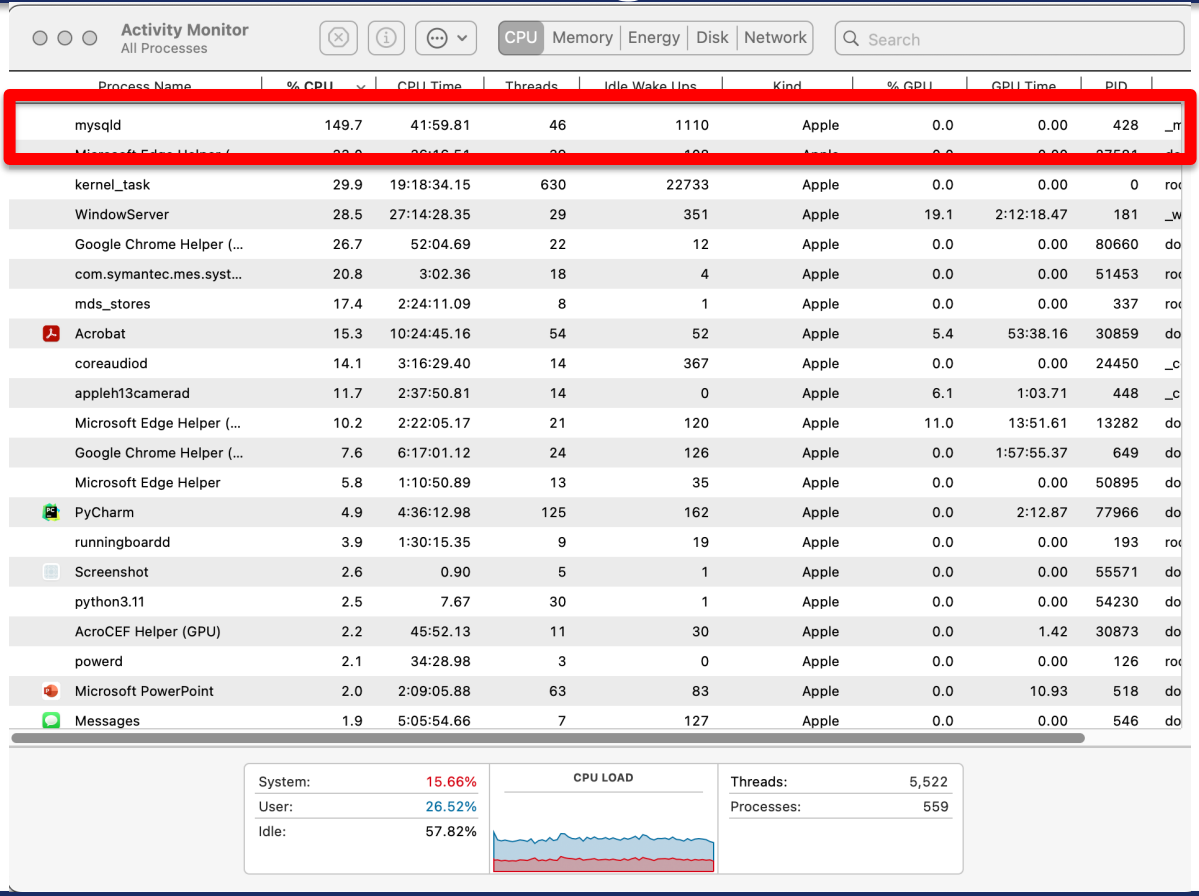
Diagram 2: The moving of the Web Layer from the Server to the Client

Walkthrough

- Simple web application template.
- Calling some cloud APIs.

Backup

Database Loading



- You can see that loading is an intensive task.
- What is happening is that there are three logical operations running in parallel
  - Read file into memory.
  - Convert CSV to SQL
  - INSERT into tables.
- Each operation is a mix of CPU and IO.