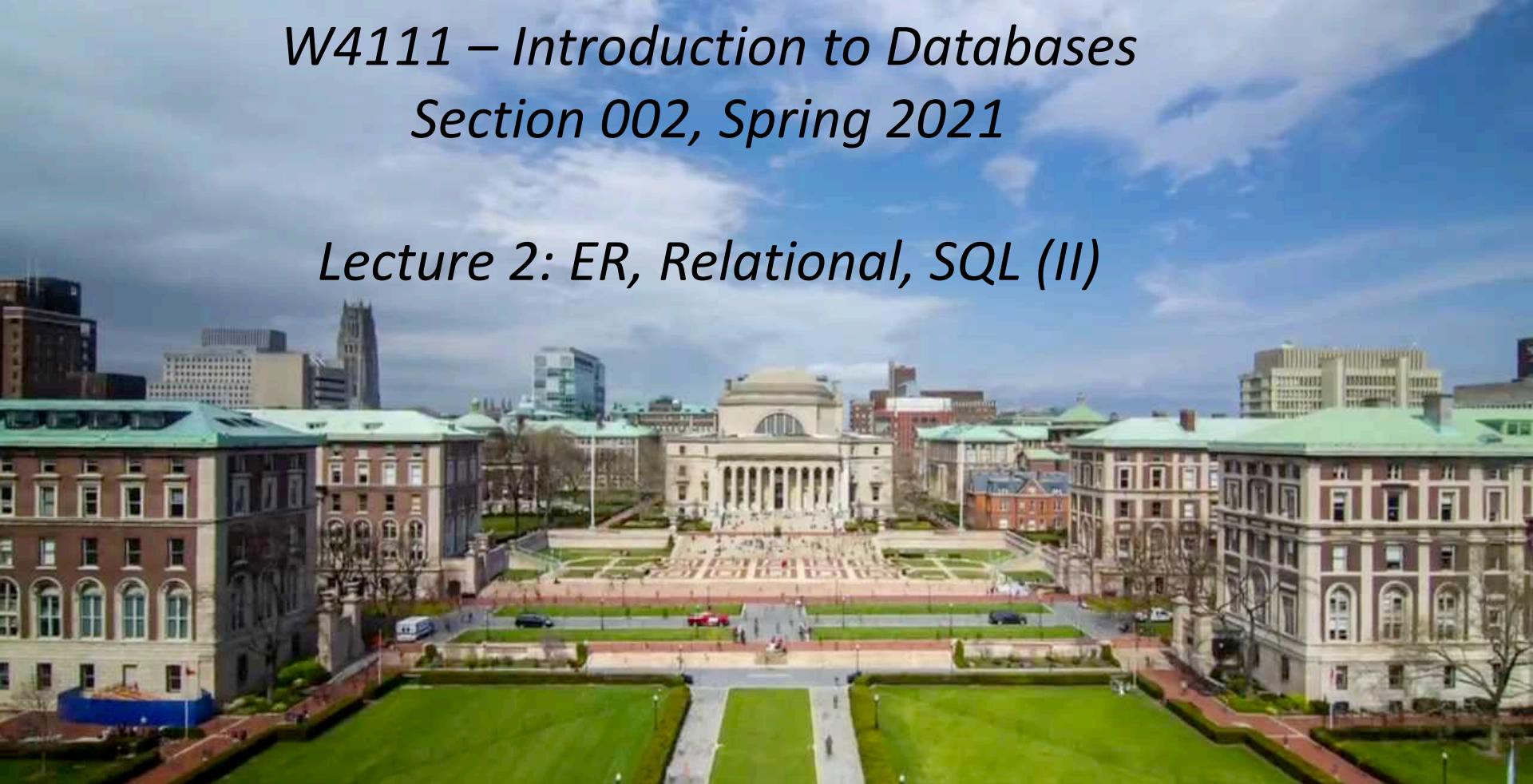


*W4111 – Introduction to Databases
Section 002, Spring 2021*

Lecture 2: ER, Relational, SQL (II)



W4111 – Introduction to Databases

Section 002, Spring 2021

Lecture 2: ER, Relational, SQL (II)

We will start in a couple of minutes.

Contents

Contents

- Introduction: comments, questions and answers
- Sample Problem: Harry Potter Characters
- ER (Diagram) Modeling (Chapter 6)
- The *Relational Model* and *Algebra* (Chapter 2)
- SQL

Questions, Answers, Comments

Computing Max Salary (Part D, Q0)

max salary

All other solutions of mine align with the solutions provided except for the max salary part. I am pretty sure I am working on the correct scheme. Is there anything wrong with my code or is it that the provided solution is just a template? Thank you.

```
In [14]: Student answer
mysql+pymysql://admin:***@jp4108-db.crm9jcopq8my.us-east-2.rds.amazonaws.com/lahmansbaseballdb
* mysql+pymysql://admin:***@jp4108-db.crm9jcopq8my.us-east-2.rds.amazonaws.com/lahmansdb_to_clean
1 rows affected.

Out[14]: Max_Salary
33000000.0

In [5]:
* mysql+pymysql://dbuser:***@localhost/lahman2019clean
1 rows affected.

Out[5]: Max_Salary
9993640
```

Which answer is correct? Well, it depends.

Sample answer

What Do You Mean, “It Depends.”

- In the past semesters, we required students to import CSV files for all tables. This semester, we simplified and have you a load script.
- In general,
 - Columns in CSV files are text for import.
 - Importing as numbers often fails because a row contains something like "", "NA", ...
- Maximum of salary.
 - If treated as text 9993640 is “bigger” than 33000000.
 - If treated as numbers 33000000 is “bigger” than 9993640.
 - For text, “>” is alphabetical order, not number value.
- Consider something like ‘value1 operator value2.’ What happens? It depends on the operators and the types.
- Language runtimes often implement *type coercion/type conversion*.

Type Coercion/Type Conversion

```
1
2 ✓ select '5' + '3';
```

Output `'5' + '3':int ×`

1	8

```
1
2 ✓ select '5' + 3;
```

Output `'5' + 3:int ×`

1	8


```
1
2 ✓ select 9 > '8999999999999999';
```

Output `9 > '8999999999999999':tinyint ×`

1	1

```
1
2 ✓ select '5' / '3';
```

Output `'5' / '3':int ×`

1	1.6666666666666667

- All languages perform coercion/conversation.
- The rules are usually similar but there are differences.

Excellent Questions

Resolved Unresolved



Christodoulos Constantines 7 minutes ago

Hello, 3 questions for data cleaning:

- 1) Do we have to add additional constraint checks for the fields? For example months must be in the range 1-12.
- 2) For the isDead field, what if new data arrives to the table? Shouldn't we add something to automatically update isDead based on the deathYear's (or any other relevant field) value?
- 3) When deathYear is null do we assume that the person is alive? What if he is dead and information is just missing?



Rich text editor Plain text editor Markdown editor

Insert Format Table



- (1) No. Not yet.
- (2) Yes, but not now. We will cover later.
- (3) How would you know he is dead? NULL means unknown/data missing or not applicable.

[post](#) [preview](#)

Screenshot

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 unknown, missing or inapplicable values.
 - String: “”, “NA”, “UNKNOWN”, ...
 - Numbers: -1, 0, -9999
- Other 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
```

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
```

avg_weight	avg_net_worth
150.0000	150.0000

Back to Our Excellent Question

- “When *deathYear* is null do we assume that the person is alive? What if he is dead and information is just missing?”
- Well,
 - My interpretation is that if there is no death date, *isDead* should be NULL.
 - I might expect that the person is dead if their current age would be 125, but that is just a guess.
- Also,
 - Having a column *isDead* is actually a bad design pattern in this case.
 - The value of *isDead* is *functionally dependent* on *deathYear*.
 - Database designs avoid *functional dependencies* to prevent *update anomalies*. Some could forget to update both columns when learning a fact.
 - In other scenarios, I might know that a person *isDead* but not know the *deathDate*. So, this design decision is problem/scenario specific.

Paraphrasing a Common Question

- There are a lot of columns. Is there anyway to
 - Figure out the list of columns.
 - Set all values with “” to NULL in one statement.
- The logic would be something like,
 - **For** c in list_of_columns(people):
 - **If** there are “” values in c **then**:
 - Update people set c=NULL where c=””
 - For, If ... Then ... else, and similar language constructs are *imperative or procedural*.
 - SQL (and the relational algebra) are *declarative*.
 - for ..., if ... then ... are specifying *how* to compute the answer.
 - SQL specifies *what* answer you want.
- So, do I have to write a lot of *update* statements?

Paraphrasing a Common Question

- There are a lot of columns. Is there anyway to
 - Figure out the list of columns.

- T

Switch to Jupyter Notebook example

procedural.

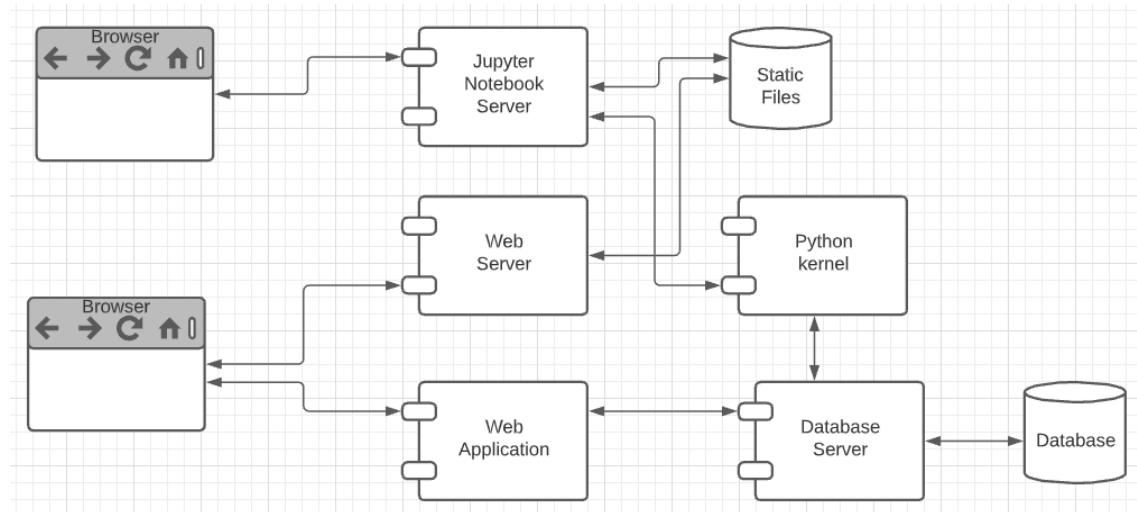
- So, do I have to write a lot of *update* statements?

Today's Task #1



Harry Potter Character Information Application

- Dataset:
 - <https://www.kaggle.com/gulsahdemiryurek/harry-potter-dataset>
 - Six “comma separated value (CSV)” files. We will start with *Characters.csv*.
- Tasks: (Both tasks start with *importing and engineering the data.*)
 - Build an interactive web application for *create-retrieve-update-delete* entries. The application must enforce *semantic integrity*.
 - Build a Jupyter Notebook that does some interesting visualization and analysis.
- Elements:
 - Content/applications in a browser.
 - Jupyter Notebook Server is both:
 - A web server
 - A (specialized) web application server.
 - Web server delivers HTML, images, ... to the browser.
 - Web application server executes an
 - Application server framework (Flask)
 - The Python code implementing the app.
 - The Jupyter Notebook Server runs code cells in a Python kernel.
 - I am mostly going to ignore the browser and content.



The Data – Characters.csv

The screenshot shows the 'Harry Potter Dataset' page on Kaggle. At the top, there's a navigation bar with 'Dataset', 'Harry Potter Dataset', 'Gulsa Demiryurek', 'updated a year ago (Version 1)', 'Data', 'Tasks', 'Notebooks (5)', 'Discussion', 'Activity', and 'Metadata'. Below this is a large preview area with a purple background featuring a grid of small Harry Potter characters. To the right of the preview are download and new notebook buttons. Underneath the preview, there are sections for 'Usability 5.3', 'Tags arts and entertainment, movies and tv shows', 'Description', 'Context' (with a note about the story behind the dataset), 'Content' (mentioning movie scripts from subtitles and data collection from pottermore.com), and 'Acknowledgements' (with a note about the source of the data). At the bottom, the 'Data Explorer' section shows the 'Characters.csv' file (25.86 KB) with a preview of its contents. The preview table has columns for Id, Name, Gender, Job, House, and Birth. The first row shows the total values: 0 total values, with counts for null (1998), 3% (Other (62)), 53% (Female), 3% (Male), 1% (Other (21)), 84% (Student), 1% (Other (5)), 94% (Gryffindor), 2% (Hufflepuff), 3% (Ravenclaw), 1% (Slytherin), and 1% (Unknown). The second row shows a specific entry for Harry James Potter.

Id	Name	Gender	Job	House	Birth
0	[null] Female Male Other (62)	53% 3% 43%	[null] Student Other (21)	84% 1% 15%	[null] Gryffindor Hufflepuff Ravenclaw Slytherin Unknown
1	Harry James Potter	Male	Student	Gryffindor	1998

15 columns:

- Id
 - Name
 - Gender
 - Job
 - House
 - Wand
 - Patronus
 - Species
 - Blood status
 - Hair colour
 - Eye colour
 - Loyalty
 - Skills
 - Birth
 - Death
- Downloaded the file.
 - The columns are separated by ‘;’
 - Despite being “Comma” Separated Values, other separators are common.
 - Loaded using DataGrip into MySQL
 - Connect
 - Create Schema
 - Use Schema
 - Use the data load tool.

But First, Let's Look at the Data

There are some weird characters, no pun intended.

- 9º" Chestnut dragon heartstring
- Dumbledore's Army | Hogwarts School of Witchcraft and Wizardry
- 9Ω" Fir dragon heartstring
- Professor of Transfiguration† | Head of Gryffindor

- How did this happen?
- What do we do?

Female	Professor of Transfiguration† Head of Gryffindor	Gryffindor	9Ω" Fir dragon heartstring
Female		Gryffindor	Unknown
Male	Head of the Misuse of Muggle Artefacts Office	Gryffindor	Unknown
Male	Defence Against the Dark Arts(1991-1992)	Ravenclaw	9" Alder unicorn hair bendy
Female	Student	Ravenclaw	Unknown
Female	Student	Ravenclaw	Unknown
Male	Defence Against the Dark Arts(1992-1993)	Ravenclaw	9" Cherry dragon heartstring
Male	Professor of Charms Head of Ravenclaw	Ravenclaw	Unknown
Female	Professor of Divination	Ravenclaw	9 Ω hazel unicorn hair core
Male	Wandmaker	Ravenclaw	12a" Hornbeam dragon heartstring
Female	Student	Ravenclaw	Unknown
Female	Student	Ravenclaw	Unknown
Male	Student	Ravenclaw	Unknown
Female	Student	Ravenclaw	Unknown
Male	Student	Ravenclaw	Unknown
Male	Student	Ravenclaw	Unknown
Male	Student	Ravenclaw	Unknown
Male	Professor of Potions Head of Slytherin	Slytherin	Unknown
Male	Student	Slytherin	10" Hawthorn unicorn hair
Male	Student	Slytherin	Unknown
Male	Student	Slytherin	Unknown
Female		Slytherin	12a" Walnut dragon heartstring
Female	Professor of Defence Against the Dark Arts† Department of Magical Law Enforcement	Slytherin	8" Birch dragon heartstring
	Professor of Potions	Slytherin	10Ω" Cedar dragon heartstring fairly flexible
Male	School Governor	Slytherin	Elm and dragon heartstring
Female		Slytherin	Unknown
Male		Slytherin	Unknown
Female	Student	Slytherin	Unknown

Analysis (I)

- Some of the issues are character encoding/set issues:
 - “In computing, data storage, and data transmission, character encoding is used to represent a repertoire of characters by some kind of encoding system that assigns a number to each character for digital representation.”
(https://en.wikipedia.org/wiki/Character_encoding)
 - “A character set is a collection of characters that might be used by multiple languages.”
(https://en.wikipedia.org/wiki/Character_encoding)
- This will be a common issue when dealing with string data that you import to build your database.
- This is especially true if the data comes from “data scraping”.
 - “Data scraping is a technique in which a computer program extracts data from human-readable output coming from another program.”
(https://en.wikipedia.org/wiki/Data_scraping)
 - There are several types of scraping: screen scraping, web scraping,
- That is what happened here. The data comes from web scraping a wiki page.

Content

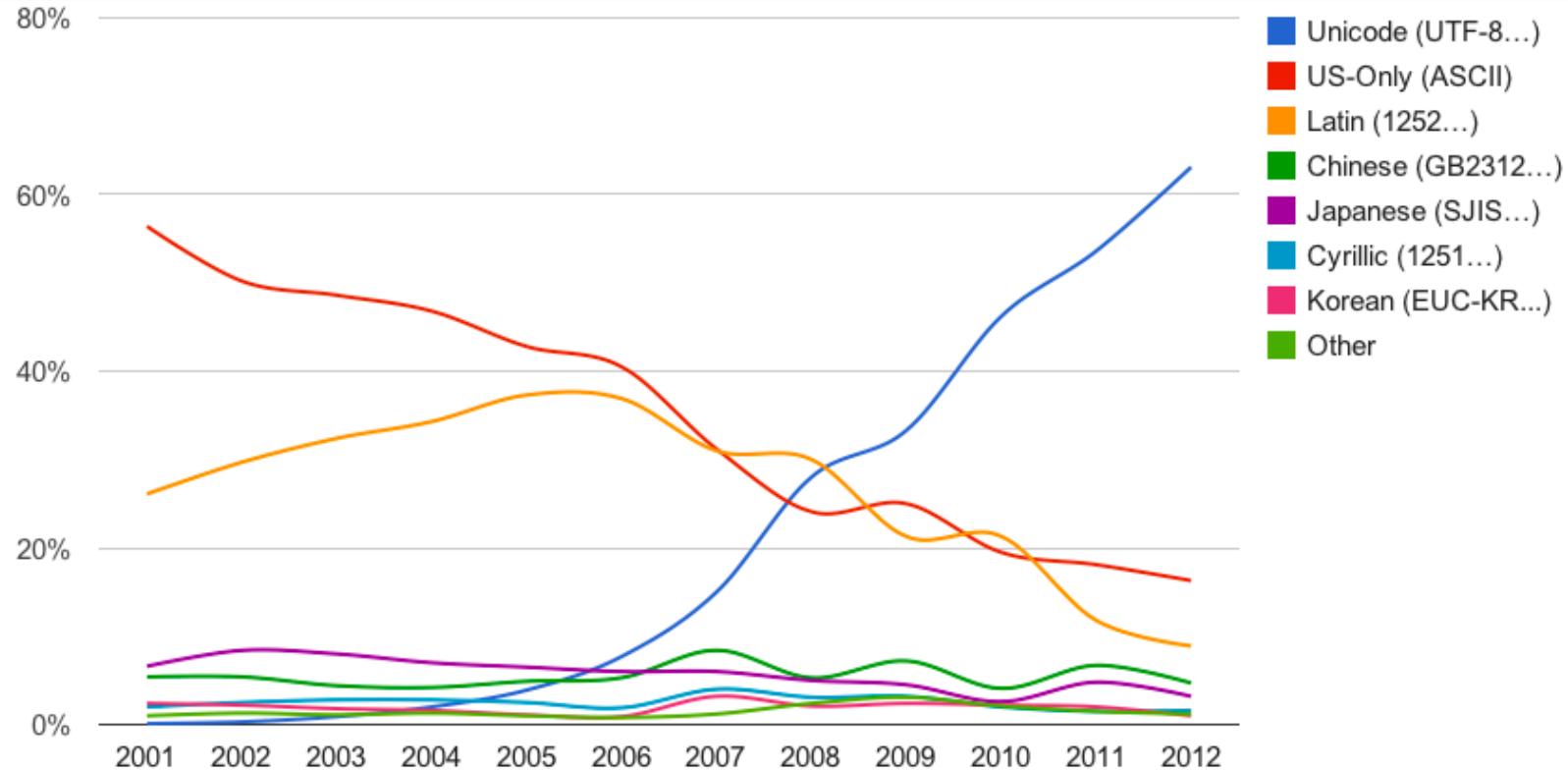
Movie scripts are from subtitles. The other data are collected from pottermore.com and https://harrypotter.fandom.com/wiki/Main_Page

Character Encodings/Character Sets

Common character encodings [edit]

- ISO 646
 - ASCII
 - EBCDIC
- ISO 8859:
 - ISO 8859-1 Western Europe
 - ISO 8859-2 Western and Central Europe
 - ISO 8859-3 Western Europe and South European (Turkish, Maltese plus Esperanto)
 - ISO 8859-4 Western Europe and Baltic countries (Lithuania, Estonia, Latvia and Lapp)
 - ISO 8859-5 Cyrillic alphabet
 - ISO 8859-6 Arabic
 - ISO 8859-7 Greek
 - ISO 8859-8 Hebrew
 - ISO 8859-9 Western Europe with amended Turkish character set
 - ISO 8859-10 Western Europe with rationalised character set for Nordic languages, including complete Icelandic set
 - ISO 8859-11 Thai
 - ISO 8859-13 Baltic languages plus Polish
 - ISO 8859-14 Celtic languages (Irish Gaelic, Scottish, Welsh)
 - ISO 8859-15 Added the Euro sign and other rationalisations to ISO 8859-1
 - ISO 8859-16 Central, Eastern and Southern European languages (Albanian, Bosnian, Croatian, Hungarian, Polish, Romanian, Serbian and Slovenian, but also French, German, Italian and Irish Gaelic)
- CP437, CP720, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872
 - MS-Windows character sets:
 - Windows-1250 for Central European languages that use Latin script, (Polish, Czech, Slovak, Hungarian, Slovene, Serbian, Croatian, Bosnian, Romanian and Albanian)
 - Windows-1251 for Cyrillic alphabets
 - Windows-1252 for Western languages
 - Windows-1253 for Greek
 - Windows-1254 for Turkish
 - Windows-1255 for Hebrew
 - Windows-1256 for Arabic
 - Windows-1257 for Baltic languages
 - Windows-1258 for Vietnamese
 - Mac OS Roman
 - KOI8-R, KOI8-U, KOI7
 - MIK
 - ISCII
 - TSCII
 - VISCII
- JIS X 0208 is a widely deployed standard for Japanese character encoding that has several encoding forms.
 - Shift JIS (Microsoft [Code page 932](#) is a dialect of Shift_JIS)
 - EUC-JP
 - ISO-2022-JP
- JIS X 0213 is an extended version of JIS X 0208.
 - Shift_JIS-2004
 - EUC-JIS-2004
 - ISO-2022-JP-2004
- Chinese Guobiao
 - GB 2312
 - GBK (Microsoft [Code page 936](#))
 - GB 18030
- Taiwan Big5 (a more famous variant is Microsoft [Code page 950](#))
 - Hong Kong HKSCS
- Korean
 - KS X 1001 is a Korean double-byte character encoding standard
 - EUC-KR
 - ISO-2022-KR
- Unicode (and subsets thereof, such as the 16-bit 'Basic Multilingual Plane')
 - UTF-8
 - UTF-16
 - UTF-32
- ANSEL or ISO/IEC 6937

Character Set Usage over Time



<https://www.w3.org/International/questions/qa-who-uses-unicode>

But,

- Consider **Job** column entry: “Professor of Transfiguration† | Head of Gryffindor”
- There are two odd characters: “†” and “|”
- How should I correct/interpret the strings? My interpretation is:
 - Replace “†” with “ ”.
 - The “|” means that the person has two jobs:
 - Professor of Transfiguration
 - Head of Gryffindor
 - The **Job** column is a **multivalued** attribute.
 - The character “|” indicates that a text string is one string containing multiple values for an attribute.
- Can I always replace non-|, odd characters with “ “?
 - ‘9Ω’ Fir dragon heartstring’ and ‘9º’ Chestnut dragon heartstring’ would wind up having an extra space. I want 9”, not 9 “.
 - The cleanup is getting complicated.

The Cleanup is Rule-Based

- In this dataset, I can use a set of rules for data clean up:
 - Two characters separated by † maps to two characters separated by “ “ (space).
 - A number and “ separated by an odd character maps the the number and “
 - Strings of the form s1 | s2 | s3 maps to [s1, s2, s3]
 - etc.
- Well, I could write a custom function for each rule,
 - But what happens when I find I need other rules.
 - It would be nice if there were some kind of tools that helps.
- A **regular expression** is a sequence of characters that define a search pattern. Usually, such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory.
(https://en.wikipedia.org/wiki/Regular_expression)

Multivalued Attributes

- How about strings of the form $s_1 | s_2 | s_3$ maps to $[s_1, s_2, s_3]$.
Is this cool?
- All the data entity sets we have seen have had simple types/value attributes.
Depends on the database model, e.g. relational, document, ...

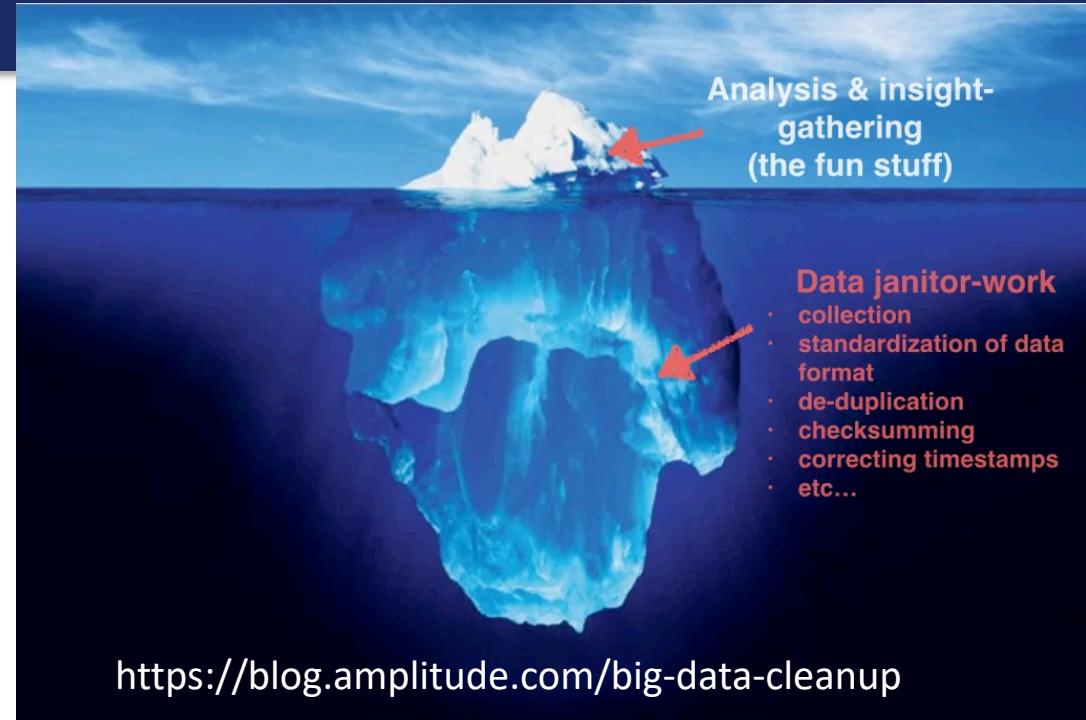
<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure 2.2 The *course* relation.

Data Cleansing



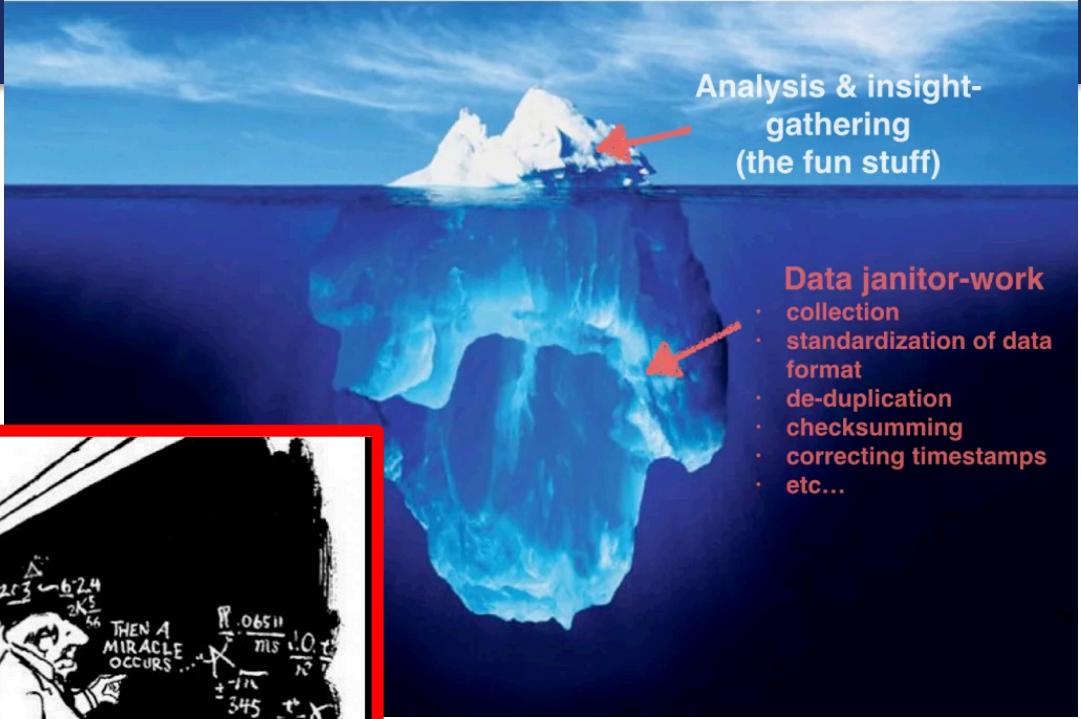
Database and data science classes **love to teach the fun stuff**
queries, data modeling, machine learning, spooky math, algorithms,

Data Cleansing

Syllabus Topics

- Relational Foundations
- Overview (1 lecture)
- ER Model (2 lectures)
- Relational Model (4 lectures)
- Relational Algebra (2 lectures)
- SQL (5 lectures)
- Application Programming and Database APIs (1 lecture)
- Security (2 lectures)
- Normalization (2 lectures)
- Overview of Storage and Indexes (1 lecture)
- Overview of Query Optimization (1 lecture)
- Overview of Transaction Processing (1 lecture)
- Beyond Relational Foundations
- NoSQL (1 lecture)
- Data Preparation and Cleaning (1 lecture)
- Graphs (1 lecture)
- Object-Relational Databases (2 lectures)
- Cloud Databases (1 lecture)

Recommended Syllabus



Analysis & insight-gathering
(the fun stuff)

Data janitor-work

- collection
- standardization of data format
- de-duplication
- checksumming
- correcting timestamps
- etc...

I place more emphasis on data cleansing and refactoring than other sections of W4111.

Next Steps in the Plan

Source Data

The screenshot shows the 'Harry Potter Dataset' on Kaggle. At the top, there's a navigation bar with 'Data', 'Tasks', 'Notebooks (1)', 'Discussion', 'Activity', 'Metadata', 'Download (93 KB)', and 'New Notebook'. Below this is a section titled 'Description' with a story about the dataset. Under 'Content', it says 'Movie scripts are from subtitles. The other data are collected from pottermore.com and https://harrypotter.fandom.com/wiki/Main_Page'. The 'Data Explorer' section shows a preview of 'Characters.csv' (256.82 KB) with columns: ID, Name, Gender, Job, House, and a detailed view of the first row for Harry James Potter.

Relational Data

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000



- We are starting the semester with the *relational datamodel/databases*.
- Before we get too far in our data janitor work, we should understand the relational model a little more.

But, Before We Leave

- What are these regular expressions whereout you speak?
- Many environments come with regular expressions functions for searching and transforming strings.
 - Python: Lib/re.py (<https://docs.python.org/3/library/re.html>)
 - Java: java.util.regex (https://www.w3schools.com/java/java_regex.asp)
 - "SNOBOL ("StriNg Oriented and symBOlic Language") is a series of programming languages having patterns as a first-class data type." (<https://en.wikipedia.org/wiki/SNOBOL>)
 - Most SQL databases have some from of regular expression library, but the libraries differ from product to product.



Table 12.14 Regular Expression Functions and Operators

Name	Description
<u>NOT_REGEXP</u>	Negation of REGEXP
<u>REGEXP</u>	Whether string matches regular expression
<u>REGEXP_INSTR()</u>	Starting index of substring matching regular expression
<u>REGEXP_LIKE()</u>	Whether string matches regular expression
<u>REGEXP_REPLACE()</u>	Replace substrings matching regular expression
<u>REGEXP_SUBSTR()</u>	Return substring matching regular expression
<u>RLIKE</u>	Whether string matches regular expression

```
In [40]: 1 s = '12æ" Hornbeam dragon heartstring'  
2 x = re.sub(r'([0-9]).(.)', r'\1\2', s)  
3 x
```

Out[40]: '12" Hornbeam dragon heartstring'

The Relational Model

The Relational Data Model



Relational Model

- All the data is stored in various tables.
- Example of tabular data in the relational model



Ted Codd
Turing Award 1981

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

- The “relation” is the “table.”
 - In my big space of pieces of data. *ID, name, dept_name, salary* are somehow related.
 - This causes confusion, because the ER and other models use “relation” to mean something else.
- Core concepts:
 - Relation
 - Tuple (Row)
 - Column (Attribute)



Example of a *Instructor* Relation

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Diagram illustrating the components of the table:

- Attributes (or columns):** Four arrows point from the text "attributes (or columns)" to the column headers *ID*, *name*, *dept_name*, and *salary*.
- Tuples (or rows):** Two arrows point from the text "tuples (or rows)" to the first two rows of the table data.



Attribute

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value **null** is a member of every domain. Indicated that the value is “unknown”
- The null value causes complications in the definition of many operations

DFF Comments:

- Atomic and use of Null is important?
- I will explain the importance of atomic attributes and null in examples.



Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000



Database Schema

- Database schema -- is the logical structure of the database.
- Database instance -- is a snapshot of the data in the database at a given instant in time.
- Example:
 - schema: *instructor (ID, name, dept_name, salary)*
 - Instance:

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000



Keys

- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
 - Example: $\{ID\}$ and $\{ID, name\}$ are both superkeys of *instructor*.
- Superkey K is a **candidate key** if K is minimal
Example: $\{ID\}$ is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
 - which one?
- **Foreign key** constraint: Value in one relation must appear in another
 - **Referencing** relation
 - **Referenced** relation
 - Example: *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

Notation

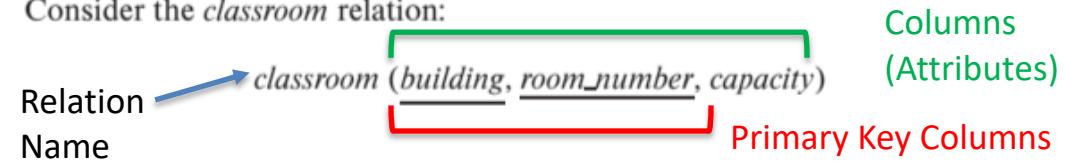
Classroom relation

building	room_number	capacity
Packard	101	500
Painter	100	125
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

classroom schema

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept_name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined.

Consider the *classroom* relation:



- The primary key is a *composite key*. Neither column is a key (unique) by itself.
- Keys are statements about all possible, valid tuples and not just the ones in the relation.
 - Capacity is unique in this specific data, but clearly not unique for all possible data.
 - In this domain, there cannot be two classrooms with the same building and room number.
- Relation schema:
 - Underline indicates a primary key column. There is no standard way to indicate other types of key.
 - We will use **bold** to indicate foreign keys.
 - You will sometimes see things like *classroom*(building:string, room_number:number, *capacity:number*)

Observations

- Keys:
 - Will be baffling. It takes time and experience to understand/appreciate.
 - There are many, many types of keys with formal definitions.
 - I explain the formality but focus on the concepts and applications.
- No one uses the formal, relational model. So, why do we study it?
 - Is very helpful when understanding concepts that we cover later in the course, especially query optimization and processing.
 - There are many realizations of the model and algebra, and understanding the foundation helps with understanding language/engine capabilities.
 - The model has helped with innovating new approaches, and you may innovate in data and query models in your future.

Relational Algebra



Relational Query Languages

- Procedural versus non-procedural, or declarative
- “Pure” languages:
 - Relational algebra
 - Tuple relational calculus
 - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this chapter on relational algebra
 - Not turning-machine equivalent
 - Consists of 6 basic operations

DFF Comments:

- You will sometimes see other operator, e.g. \leftarrow Assignment.
- Relational algebra focuses on *retrieve*. You can sort of do Create, Update, Delete.
- The SQL Language, which we will see, extends relational algebra.



Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
 - select: σ
 - project: Π
 - union: \cup
 - set difference: $-$
 - Cartesian product: \times
 - rename: ρ



Select Operation

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Example: select those tuples of the *instructor* relation where the instructor is in the “Physics” department.
 - Query

$$\sigma_{dept_name = "Physics"}(instructor)$$

- Result

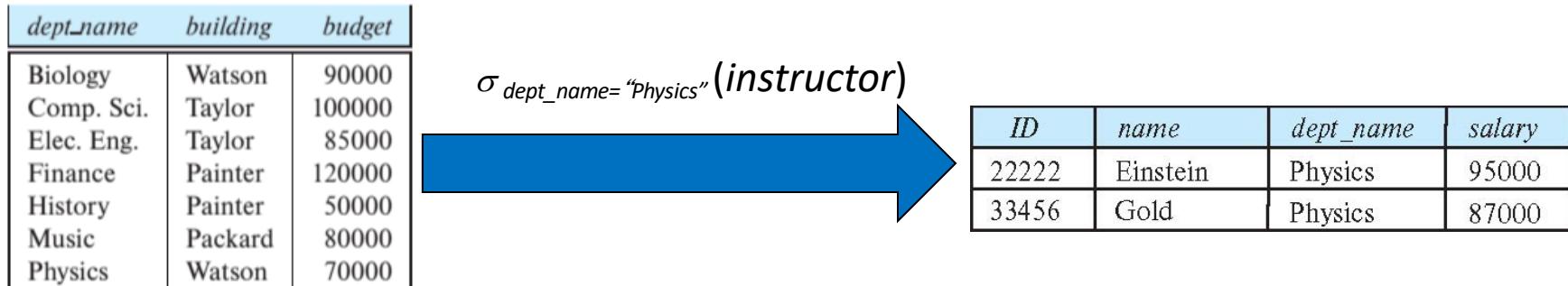


Figure 2.5 The *department* relation.



Select Operation (Cont.)

- We allow comparisons using
 $=, \neq, >, \geq, <, \leq$
in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:
 \wedge (**and**), \vee (**or**), \neg (**not**)
- Example: Find the instructors in Physics with a salary greater \$90,000, we write:

$$\sigma_{dept_name = "Physics"} \wedge salary > 90,000 (instructor)$$

- Then select predicate may include comparisons between two attributes.
 - Example, find all departments whose name is the same as their building name:
 - $\sigma_{dept_name=building} (department)$



Project Operation

- A unary operation that returns its argument relation, with certain attributes left out.
- Notation:

$$\Pi_{A_1, A_2, A_3, \dots, A_k} (r)$$

where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets



Project Operation (Cont.)

- Example: eliminate the *dept_name* attribute of *instructor*
- Query:

$$\Pi_{ID, name, salary} (instructor)$$

- Result:

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

$$\Pi_{ID, name, salary} (instructor)$$


<i>ID</i>	<i>name</i>	<i>salary</i>
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000



Composition of Relational Operations

- The result of a relational-algebra operation is relation and therefore of relational-algebra operations can be composed together into a **relational-algebra expression**.
- Consider the query -- Find the names of all instructors in the Physics department.

$$\Pi_{name}(\sigma_{dept_name = "Physics"}(instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

Relax Calculator

- Let's look at an online tool that you will use.
- RelaX (<https://dbis-uibk.github.io/relax/calc/local/uibk/local/0>)
- The calculator:
 - Does have the sample data from the textbook.
 - You can also upload new data. (Show how and example)
- Some queries:
 - $\sigma \text{ title='Computer Science Department'} \vee \text{title='Accounting Division'} \text{ (departments)}$
 - $\pi \text{ last_name, email } \text{ (faculty)}$
 - $\pi \text{ last_name, email } ($
 $\sigma \text{ department_id=165} \vee \text{department_id=1010 } \text{ (faculty)}$
)

SQL





You can think of SQL being
an extended, usable, useful version
of the relational model.

Chapter 3: Introduction to SQL

Database System Concepts, 7th Ed.

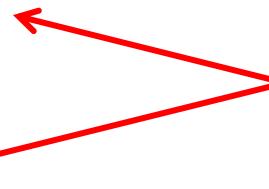
©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use



Outline

- Overview of The SQL Query Language
- SQL Data Definition
- Basic Query Structure of SQL Queries
- ~~Additional Basic Operations~~
- ~~Set Operations~~
- Null Values
- ~~Aggregate Functions~~
- ~~Nested Subqueries~~
- Modification of the Database

Will cover in next lecture
and/or tutorial.





History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
 - SQL-86
 - SQL-89
 - SQL-92
 - SQL:1999 (language name became Y2K compliant!)
 - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
 - Not all examples here may work on your particular system.



SQL Parts

- DML -- provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- integrity – the DDL includes commands for specifying integrity constraints.
- View definition -- The DDL includes commands for defining views.
- Transaction control –includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and dynamic SQL -- define how SQL statements can be embedded within general-purpose programming languages.
- Authorization – includes commands for specifying access rights to relations and views.



Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.



Domain Types in SQL

- **char(*n*)**. Fixed length character string, with user-specified length *n*.
- **varchar(*n*)**. Variable length character strings, with user-specified maximum length *n*.
- **int**. Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **numeric(*p,d*)**. Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric(3,1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(*n*)**. Floating point number, with user-specified precision of at least *n* digits.
- More are covered in Chapter 4.



Create Table Construct

- An SQL relation is defined using the **create table** command:

create table *r*

$(A_1 D_1, A_2 D_2, \dots, A_n D_n,$
 $\text{(integrity-constraint}_1\text{)},$
 $\dots,$
 $\text{(integrity-constraint}_k\text{)})$

- r* is the name of the relation
 - each A_i is an attribute name in the schema of relation *r*
 - D_i is the data type of values in the domain of attribute A_i
- Example:

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



Integrity Constraints in Create Table

- Types of integrity constraints
 - **primary key** (A_1, \dots, A_n)
 - **foreign key** (A_m, \dots, A_n) **references** r
 - **not null**
- SQL prevents any update to the database that violates an integrity constraint.
- Example:

```
create table instructor (
    ID          char(5),
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department);
```



And a Few More Relation Definitions

- ```
create table student (
 ID varchar(5),
 name varchar(20) not null,
 dept_name varchar(20),
 tot_cred numeric(3,0),
 primary key (ID),
 foreign key (dept_name) references department);
```
  
- ```
create table takes (
    ID          varchar(5),
    course_id   varchar(8),
    sec_id      varchar(8),
    semester    varchar(6),
    year        numeric(4,0),
    grade       varchar(2),
    primary key (ID, course_id, sec_id, semester, year) ,
    foreign key (ID) references student,
    foreign key (course_id, sec_id, semester, year) references section);
```



And more still

- **create table** *course* (
 course_id **varchar**(8),
 title **varchar**(50),
 dept_name **varchar**(20),
 credits **numeric**(2,0),
 primary key (*course_id*),
 foreign key (*dept_name*) **references** *department*);



Updates to tables

- **Insert**
 - `insert into instructor values ('10211', 'Smith', 'Biology', 66000);`
- **Delete**
 - Remove all tuples from the *student* relation
 - `delete from student`
- **Drop Table**
 - `drop table r`
- **Alter**
 - `alter table r add A D`
 - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.
 - All existing tuples in the relation are assigned *null* as the value for the new attribute.
 - `alter table r drop A`
 - where *A* is the name of an attribute of relation *r*
 - Dropping of attributes not supported by many databases.



Basic Query Structure

- A typical SQL query has the form:

```
select  $A_1, A_2, \dots, A_n$ 
  from  $r_1, r_2, \dots, r_m$ 
    where  $P$ 
```

- A_i represents an attribute
- R_i represents a relation
- P is a predicate.
- The result of an SQL query is a relation.

Note:

- The SELECT ... FROM ... WHERE ... Combines two relational operators, σ and Π .
- Actually, it also combines other operators, e.g. \times



The select Clause

- The **select** clause lists the attributes desired in the result of a query
 - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

```
select name  
      from instructor
```
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
 - E.g., $Name \equiv NAME \equiv name$
 - Some people use upper case wherever we use bold font.



The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after `select`.
- Find the department names of all instructors, and remove duplicates

```
select distinct dept_name  
from instructor
```

- The keyword **all** specifies that duplicates should not be removed.

```
select all dept_name  
from instructor
```



The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

```
select *
from instructor
```

- An attribute can be a literal with no **from** clause

```
select '437'
```

- Results is a table with one column and a single row with value “437”
- Can give the column a name using:

```
select '437' as FOO
```

- An attribute can be a literal with **from** clause

```
select 'A'
from instructor
```

- Result is a table with one column and N rows (number of tuples in the *instructors* table), each row with value “A”



The select Clause (Cont.)

- The **select** clause can contain arithmetic expressions involving the operation, $+$, $-$, $*$, and $/$, and operating on constants or attributes of tuples.
 - The query:

```
select ID, name, salary/12  
from instructor
```

would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

- Can rename “*salary/12*” using the **as** clause:

```
select ID, name, salary/12 as monthly_salary
```



The where Clause

- The **where** clause specifies conditions that the result must satisfy
 - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

```
select name  
from instructor  
where dept_name = 'Comp. Sci.'
```

- SQL allows the use of the logical connectives **and**, **or**, and **not**
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and \neq .
- Comparisons can be applied to results of arithmetic expressions
- To find all instructors in Comp. Sci. dept with salary > 80000

```
select name  
from instructor  
where dept_name = 'Comp. Sci.' and salary > 80000
```



The from Clause

- The **from** clause lists the relations involved in the query
 - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

```
select *
  from instructor, teaches
```

 - generates every possible instructor – teaches pair, with all attributes from both relations.
 - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).



Examples

- Find the names of all instructors who have taught some course and the course_id
 - **select** *name, course_id*
from *instructor , teaches*
where *instructor.ID = teaches.ID*
- Find the names of all instructors in the Art department who have taught some course and the course_id
 - **select** *name, course_id*
from *instructor , teaches*
where *instructor.ID = teaches.ID and instructor.dept_name = 'Art'*



The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

old-name as new-name

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

- **select distinct** *T.name*
from *instructor as T, instructor as S*
where *T.salary > S.salary and S.dept_name = 'Comp. Sci.'*

- Keyword **as** is optional and may be omitted

instructor as T \equiv *instructor T*



Null Values

- It is possible for tuples to have a null value, denoted by **null**, for some of their attributes
- **null** signifies an **unknown value** or that a **value does not exist**.
- The result of any arithmetic expression involving **null** is **null**
 - Example: $5 + \text{null}$ returns **null**
- The predicate **is null** can be used to check for null values.
 - Example: Find all instructors whose salary is null.

```
select name  
from instructor  
where salary is null
```

- The predicate **is not null** succeeds if the value on which it is applied is not null.

Note:

- **NULL is an extremely important concept.**
- **You will find it hard to understand for a while.**



Null Values (Cont.)

- SQL treats as **unknown** the result of any comparison involving a null value (other than predicates **is null** and **is not null**).
 - Example: $5 < \text{null}$ or $\text{null} \neq \text{null}$ or $\text{null} = \text{null}$
- The predicate in a **where** clause can involve Boolean operations (**and**, **or**, **not**); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
 - **and** : $(\text{true and unknown}) = \text{unknown}$,
 $(\text{false and unknown}) = \text{false}$,
 $(\text{unknown and unknown}) = \text{unknown}$
 - **or**: $(\text{unknown or true}) = \text{true}$,
 $(\text{unknown or false}) = \text{unknown}$
 $(\text{unknown or unknown}) = \text{unknown}$
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*



Modification of the Database

- Deletion of tuples from a given relation.
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation



Deletion

- Delete all instructors

delete from *instructor*

- Delete all instructors from the Finance department

delete from *instructor*
where *dept_name*= 'Finance';

- *Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building.*

delete from *instructor*
where *dept_name* **in** (**select** *dept_name*
from *department*
where *building* = 'Watson');



Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

```
delete from instructor  
where salary < (select avg (salary)  
         from instructor);
```

- Problem: as we delete tuples from *instructor*, the average salary changes
- Solution used in SQL:
 1. First, compute **avg** (*salary*) and find all tuples to delete
 2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)



Insertion

- Add a new tuple to *course*

```
insert into course
```

```
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- or equivalently

```
insert into course (course_id, title, dept_name, credits)
```

```
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Add a new tuple to *student* with *tot_creds* set to null

```
insert into student
```

```
values ('3003', 'Green', 'Finance', null);
```



Insertion (Cont.)

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of \$18,000.

```
insert into instructor
    select ID, name, dept_name, 18000
        from student
    where dept_name = 'Music' and total_cred > 144;
```

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation.

Otherwise queries like

```
insert into table1 select * from table1
```

would cause problem



Updates

- Give a 5% salary raise to all instructors

```
update instructor  
    set salary = salary * 1.05
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor  
    set salary = salary * 1.05  
    where salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor  
    set salary = salary * 1.05  
    where salary < (select avg (salary)  
                    from instructor);
```



Updates (Cont.)

- Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others by a 5%
 - Write two **update** statements:

```
update instructor
  set salary = salary * 1.03
  where salary > 100000;
update instructor
  set salary = salary * 1.05
  where salary <= 100000;
```

- The order is important
- Can be done better using the **case** statement (next slide)



Case Statement for Conditional Updates

- Same query as before but with case statement

```
update instructor  
set salary = case  
    when salary <= 100000 then salary * 1.05  
    else salary * 1.03  
end
```



Updates with Scalar Subqueries

- Recompute and update tot_creds value for all students

```
update student S
set tot_cred = (select sum(credits)
                 from takes, course
                where takes.course_id = course.course_id and
                      S.ID= takes.ID.and
                           takes.grade <> 'F' and
                           takes.grade is not null);
```

- Sets tot_creds to null for students who have not taken any course
- Instead of **sum(credits)**, use:

```
case
    when sum(credits) is not null then sum(credits)
    else 0
end
```

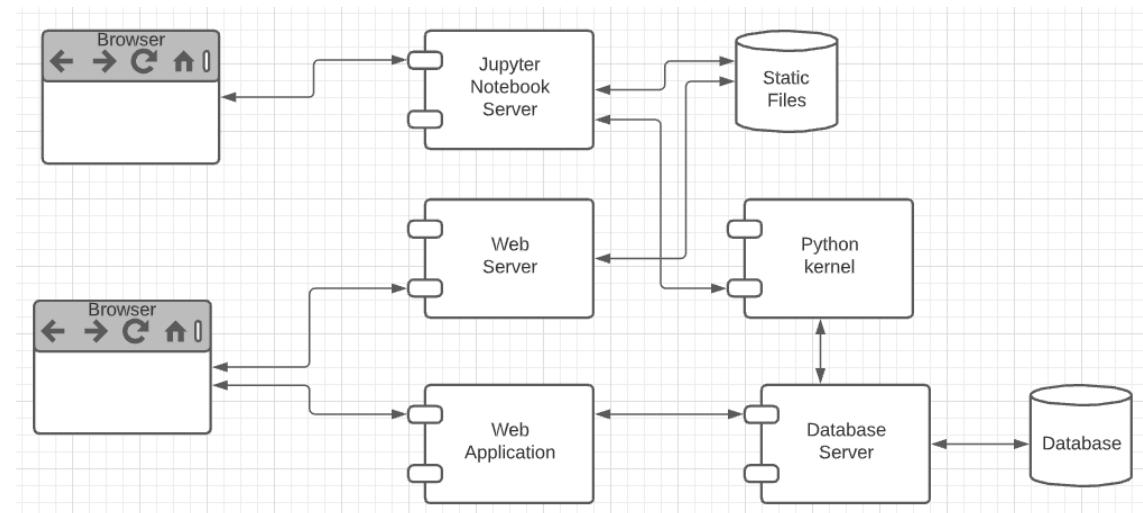
Today's Task #1

*I did the set up and provided the tools,
but did not do the task.*

I will cover in a recitation.

Harry Potter Character Information Application (Reminder)

- Dataset:
 - <https://www.kaggle.com/gulsahdemiryurek/harry-potter-dataset>
 - Six “comma separated value (CSV)” files. We will start with *Characters.csv*.
- Tasks: (Both tasks start with *importing and engineering the data.*)
 - Build an interactive web application for *create-retrieve-update-delete* entries. The application must enforce *semantic integrity*.
 - Build a Jupyter Notebook that does some interesting visualization and analysis.
- Elements:
 - Content/applications in a browser.
 - Jupyter Notebook Server is both:
 - A web server
 - A (specialized) web application server.
 - Web server delivers HTML, images, ... to the browser.
 - Web application server executes an
 - Application server framework (Flask)
 - The Python code implementing the app.
 - The Jupyter Notebook Server runs code cells in a Python kernel.
 - I am mostly going to ignore the browser and content.



Next Steps in the Plan

Source Data

The screenshot shows the 'Harry Potter Dataset' page on Kaggle. It includes a summary section with the dataset's name, size (256.82 KB), and a preview of the data. Below this is a 'Data Explorer' sidebar listing other CSV files such as 'Characters.csv', 'Spells.csv', and 'Potions.csv'. The main area displays a preview of the 'Characters.csv' file.



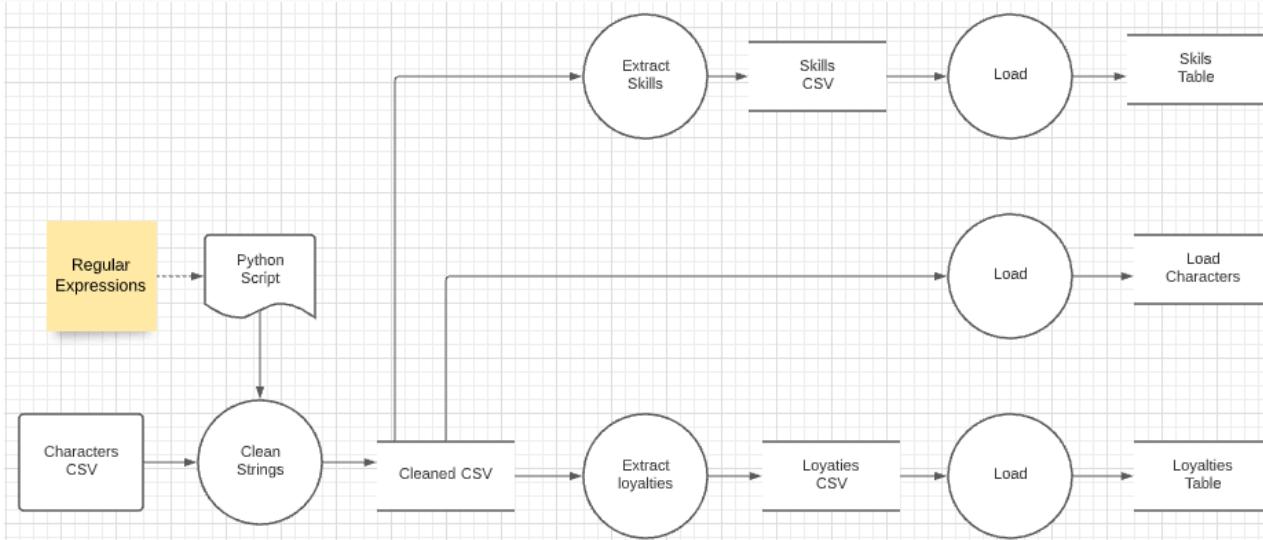
Relational Data

A table representing relational data with columns: ID, name, dept_name, and salary. Annotations show 'attributes (or columns)' pointing to the headers and 'tuples (or rows)' pointing to the data rows.

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

- We are starting the semester with the *relational datamodel/databases*.
- Before we get too far in our data janitor work, we should understand the relational model a little more.

Data Flow



- I downloaded the Harry Potter character set CSV file.
- I had two cleanup problems.
 - The file contains some weird characters in strings, e.g. "9Ω" Fir dragon heartstring
 - There is a one-to-many relationship in columns marked with delimiters, e.g. Professor of Transfiguration | Head of Gryffindor
- I wrote a little Extract-Transform-Load (ETL) process.
We will cover ETL in Module IV in the final part of the semester.

The next steps are to clean up the data in the RDB.