Introduction to Data Engineering

June 26, 2018

Course Outline

- **Relational databases**
  - Relational model
  - Relationships
  - Constraints
  - Indexing
  - Stored procedures
  - Normalization
- **SQL**
  - History / Alternatives
  - Design
  - Syntax
- **Data Marts**
  - OLAP Vs. OLTP
  - Star schema
  - Snowflake schema
- **Tidy data**
  - Principles
  - Benefits
  - Tidying messy data

Data Science Roles

ROLES	RESPONSIBILITES
Data Architect	Develops data architecture to effectively capture, integrate, organize, centralize and maintain data. Core responsibilities include: ✔ Data Warehousing Solutions ✔ Extraction, Transformation and Load (ETL) ✔ Data Architecture Development ✔ Data Modeling
Data Engineer	Develop, test and maintain data architectures to keep data accessible and ready for analysis. Key tasks are: ✔ Extraction Transformation and Load (ETL) ✔ Installing Data Warehousing Solutions ✔ Data Modeling ✔ Data Architecture Construction and Development ✔ Database Architecture Testing
Data Analyst	Processes and interprets data to get actionable insights for a company. Responsibilities include: ✔ Data Collection and Processing ✔ Programming ✔ Machine Learning ✔ Data Munging ✔ Data Visualization ✔ Applying Statistical Analysis
Data Scientist	Data analysis once data volume and velocity reaches a level requiring sophisticated technical skills. Core tasks are: ✔ Data Cleansing and Processing ✔ Predictive Modeling ✔ Machine Learning ✔ Identifying Questions ✔ Running Queries ✔ Applying Statistical Analysis ✔ Correlating Disparate Data ✔ Storytelling and Visualization

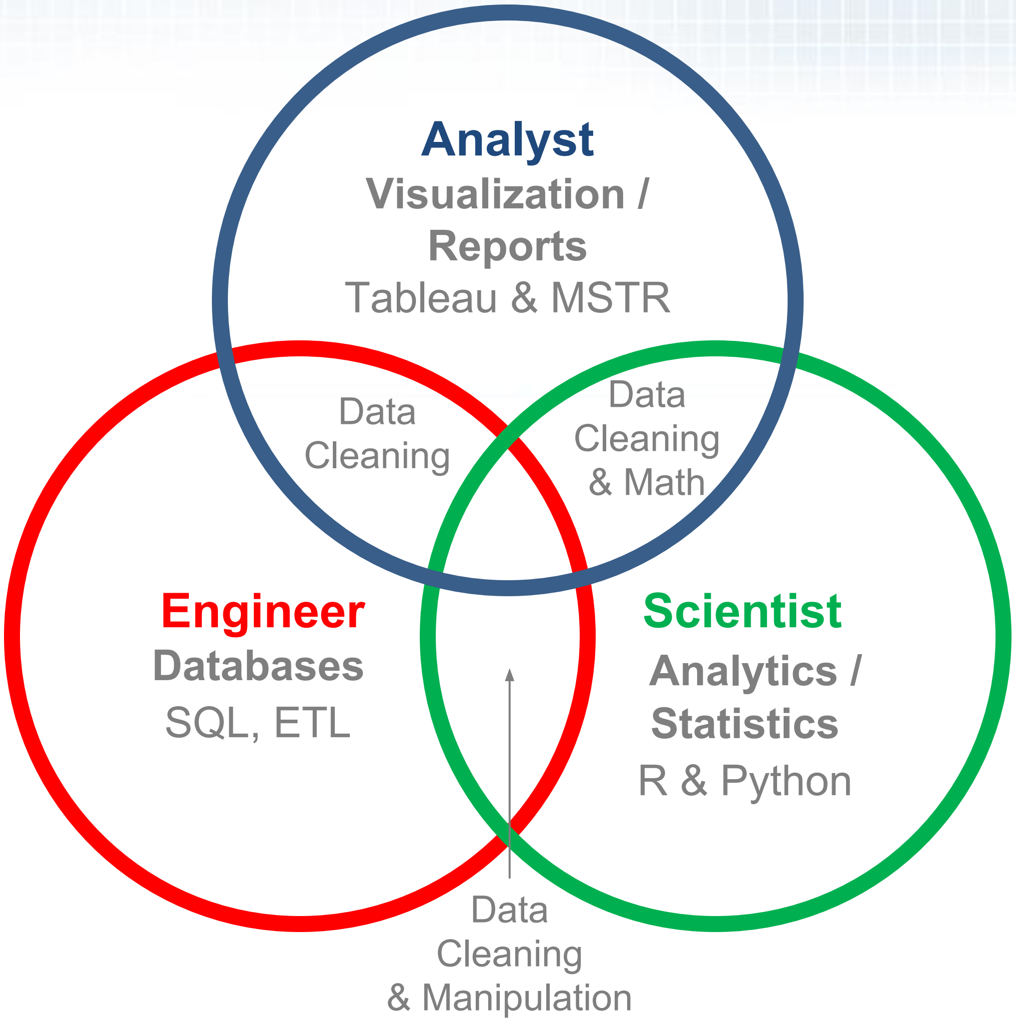
Sources:
KDnuggets – www.kdnuggets.com/2015/11/different-data-science-roles-industry.html
Udacity – blog.udacity.com/2014/12/data-analyst-vs-data-scientist-vs-data-engineer.html
RJMetrics – rjmetrics.com/resources/reports/the-state-of-data-science/

BetterBuys

GSA

D2D
DATA TO DECISIONS

D2D Data Science Roles and Tools



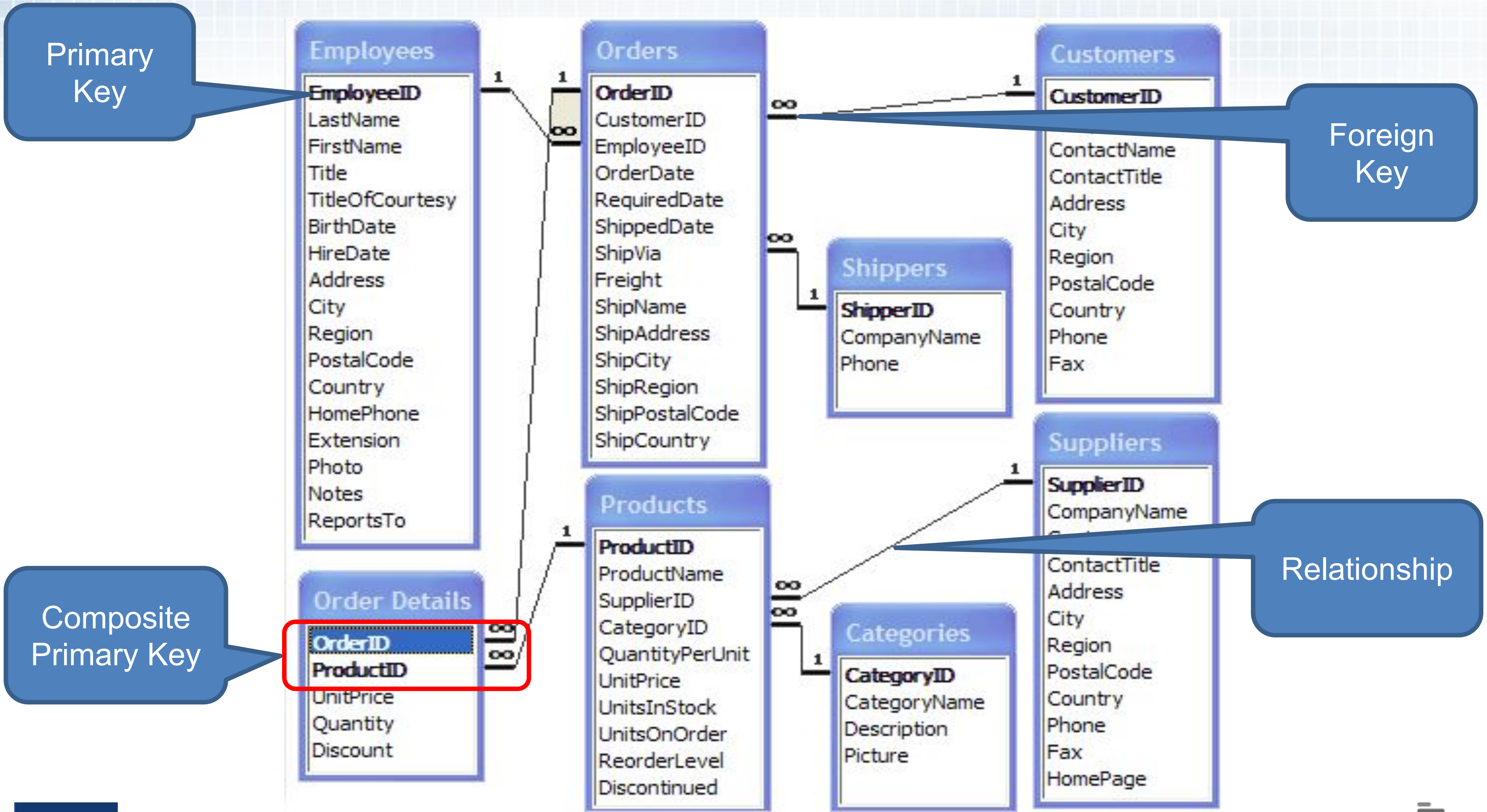
Relational Databases

- Relational databases like MySQL, PostgreSQL and SQLite3 represent and store data in tables and rows.
- Relational databases use Structured Querying Language (SQL)
  - Good for applications that involve the management of several transactions
- The structure of a relational database allows you to link information from different tables through the use of foreign keys, which are used to uniquely identify any atomic piece of data within that table.
- Other tables may refer to that foreign key, so as to create a link between their data pieces and the piece pointed to by the foreign key.
  - Comes in handy for applications that are heavy into data analysis.

Relational Databases (Cont.)

- A relational database at its simplest is a set of tables used for storing data. Each table has a unique name and may relate to one or more other tables in the database through common values.
- A table in a database is a collection of rows and columns. Tables are also known as entities or relations.
- A row contains data pertaining to a single item or record in a table. Rows are also known as records or tuples.
- A column contains data representing a specific characteristic of the records in the table. Columns are also known as fields or attributes.
- A relationship is a link between two tables (i.e., relations). Relationships make it possible to find data in one table that pertains to a specific record in another table.

Example of Relational Database



Datatypes

- Each of a table's columns has a defined datatype that specifies the type of data that can exist in that column, for example:
  - String
    - Variable Character (can define character set, i.e. ASCII)
    - BLOB (Binary Large Object)
    - Computer code, e.g. JSON
    - Text, etc.
  - Numeric can be in a form of
    - Integer (small, big, medium)
    - Double (fixed, floating)
    - Large Numeric
  - Logical
    - Boolean
  - Various formats for date and time
- Unfortunately, datatypes vary widely between databases and analytical tools

Operators

Operator Description Example

\+ Addition - Adds values on either side of the operator a + b Subtraction - Subtracts right hand operand from left hand operand a - b
* Multiplication - Multiplies values on either side of the operator a * b
/ Division - Divides left hand operand by right hand operand b / a
% Modulus - Divides left hand operand by right hand operand and returns remainder b % a

Constraints

- Constraints are used to specify rules for the data in a table
- Constraints are used to limit the type of data that can go into a table to ensure the accuracy and reliability
- Constraints can be column level or table level
- The commonly used constraints are:
  - NOT NULL - Ensures that a column cannot have a NULL value
  - UNIQUE - Ensures that all values in a column are different
  - PRIMARY KEY - A combination of a NOT NULL and UNIQUE Uniquely identifies each row in a table
  - FOREIGN KEY - Uniquely identifies a row/record in another table
  - CHECK - Ensures that all values in a column satisfies a specific condition
  - DEFAULT - Sets a default value for a column when no value is specified
  - INDEX - Used to create and retrieve data from the database very quickly

GSA

D2D

DATA TO DECISIONS

Data Integrity

The following categories of data integrity exist with each RDBMS:

- Entity Integrity: There are no duplicate rows in a table.
- Domain Integrity: Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- Referential integrity: Rows cannot be deleted, which are used by other records.
- User-Defined Integrity: Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Indexing

- Indexes are used to retrieve data from the database very fast
- The users cannot see the indexes, they are just used to speed up searches/queries
- Updating a table with indexes takes more time than updating a table without (because the indexes also need an update)
- Create indexes on columns that will be frequently searched against!
- Indexes can be unique or not unique
  - Recommend unique indexes

GSA

D2D
DATA TO DECISIONS

Database Normalization

- The concept of Normalization was introduced in 1969 by Edgar F. Codd as an integral part of his relational model
- Basic objective was to permit data to be queried and manipulated using a "universal data sub-language" grounded in first-order logic ("If X is Socrates and X is a man, then Socrates is a man")
- Has multiple states / forms
- Objectives of First Normal Form
  - Free the collection of relations from undesirable insertion, update and deletion dependencies
  - Reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs
  - Make the relational model more informative to users
  - Make the collection of relations neutral to the query statistics, i.e. query performance measurements

GSA

D2D
DATA TO DECISIONS

Database Normalization Example

Customer Name	Customer Address	Customer Tel No.	Product Name	Unit Cost	Quantity	Total Cost
Alex Wilson	1318 Scenic Avenue, Bothel	697-555-0142	Men's Sports Shorts, S	15.5	2	31
Alex Wilson	1318 Scenic Avenue, Bothel	697-555-0142	Water Bottle - 30 oz	1.5	3	4.5
Alex Wilson	1318 Scenic Avenue, Bothel	697-555-0142	LL Mountain Handlebars	19.76	2	39.52
Emily Brown	628 Muir Road, Los Angeles	708-555-0141	Long-Sleeve Logo Jersey, S	38.49	1	38.49
Emily Brown	628 Muir Road, Los Angeles	708-555-0141	Sport-100 Helmet, Black	13.08	2	26.16
Emily Brown	628 Muir Road, Los Angeles	708-555-0141	LL Mountain Handlebars	19.76	3	59.28

Question: Does it go far enough?
What if I want to know customers in same city?

Customers

Customer ID	Customer Name	Customer address	Customer Tel No.
25	Alex Wilson	1318 Scenic Avenue, Bothel	697-555-0142
48	Emily Brown	628 Muir Road, Los Angeles	708-555-0141

Orders

Order ID	Customer ID
ORD002	25
ORD004	25
ORD009	25
ORD100	48
ORD110	48
ORD125	48

Products

Product ID	Product Name	Unit Cost
P003	LL Mountain Handlebars	19.76
P020	Long-Sleeve Logo Jersey, S	38.49
P065	Sport-100 Helmet, Black	13.08
P080	Men's Sports Shorts, S	15.5
P085	Water Bottle - 30 oz.	1.5

Order Details

Order ID	Product ID	Quantity
ORD002	P080	2
ORD004	P085	3
ORD009	P003	2
ORD100	P020	1
ORD110	P065	2
ORD125	P003	3

GSA

D2D
DATA TO DECISIONS

Popular Databases

- **Commercial**
  - Oracle is the most popular relational database. It runs on both Unix and Windows. It used to be many times more expensive than SQL Server and DB2, but it has come down a lot in price.
  - SQL Server is Microsoft's database and, not surprisingly, only runs on Windows. It has only a slightly higher market share than Oracle on Windows machines. Many people find it easier to use than Oracle.
  - IBM's DB2 was one of the earliest players in the database market. It is still very commonly used on mainframes and runs on both Windows and Unix.
- **Popular Open Source Databases**
  - Until recently, PostgreSQL was the most popular open source database until that spot was taken over by MySQL. It is certainly a featureful and robust database management system and a good choice for people who want some of the advanced features that MySQL doesn't yet have.
  - Because of its small size, its speediness, and its very good documentation, MySQL has quickly become the most popular open source database. MySQL is available on both Windows and Unix. It catches up with PostgreSQL functionality.
  - DSVD is using MySQL and MS SQL Server

Brief History of SQL

Structured Query Language

- In 1970, E. F. Codd published "A Relational Model of Data for Large Shared Data Banks," an article that outlined a model for storing and manipulating data using tables
- Shortly after, IBM began working on creating a relational database
- Between 1979 and 1982, Oracle (then Relational Software, Inc.), Relational Technology, Inc. (later acquired by Computer Associates), and IBM all put out commercial relational databases
- By 1986 they all were using SQL as the data query language.
- In 1986, the American National Standards Institute (ANSI) standardized SQL
  - This standard was updated in 1989, in 1992 (called SQL2)
  - In 1999 called SQL3
  - In 2003 called SQL 2003
  - In 2006 called SQL 2006
  - In 2008 called SQL 2008
- Standard SQL is sometimes called ANSI SQL. All major relational databases support this standard but each has its own proprietary extensions

GSA

D2D
DATA TO DECISIONS

SQL Statements

- **Database Manipulation Language (DML) statements are used to work with data in an existing database. The most common DML statements are:**
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- **Database Definition Language (DDL) statements are used to structure objects in a database. The most common DDL statements are:**
  - CREATE
  - ALTER
  - DROP
- **Database Control Language (DCL) statements are used for database administration. The most common DCL statements are:**
  - GRANT
  - DENY (SQL Server Only)
  - REVOKE

Some Basics

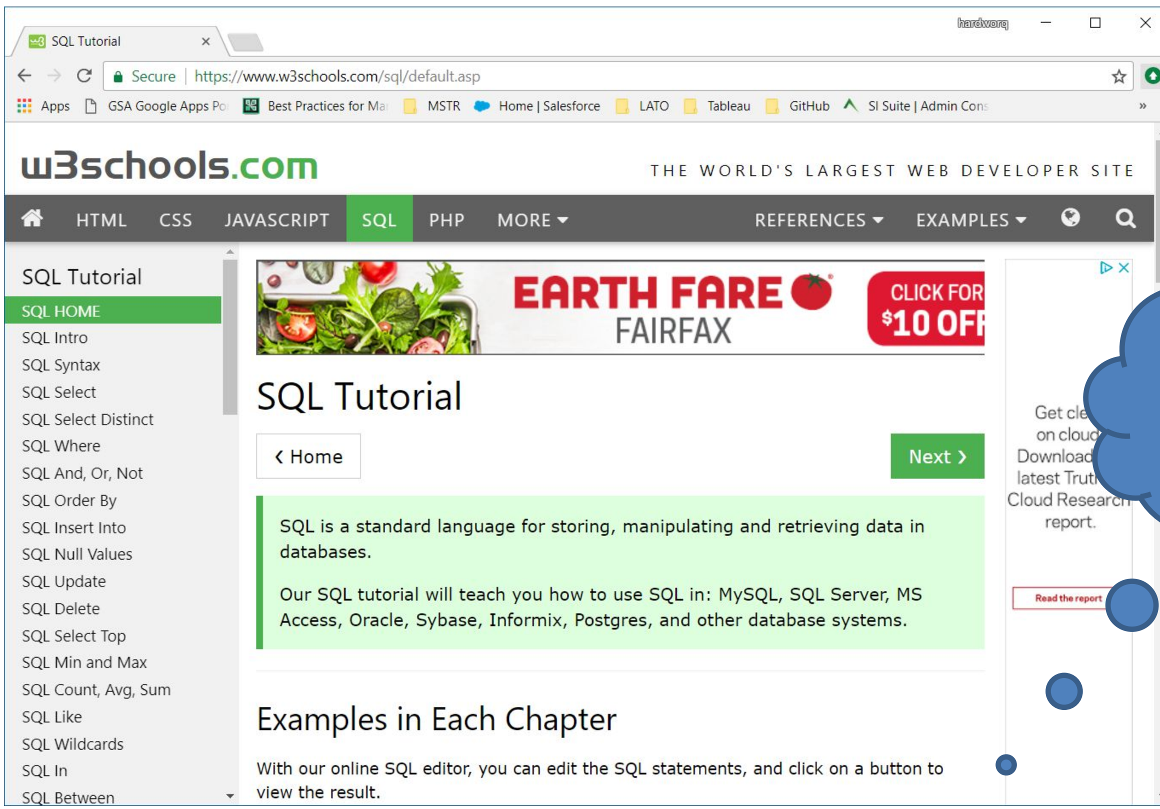
- Comments: the standard SQL comment is two hyphens (--). However, some databases use other forms of comments as shown in the table below.

Example	-- Comment	# Comment	/* Comment */
ANSI	YES	NO	NO
SQL Server	YES	NO	YES
Oracle	YES	NO	YES
MySQL	YES	YES	YES

- Whitespace is ignored in SQL statements. Multiple statements are separated with semi-colons. The two statements in the sample below are equally valid.
  - SELECT * FROM Employees;
  - SELECT *
    FROM Employees;
- SQL is not case sensitive. It is common practice to write reserved words in all capital letters. User-defined names, such as table names and column names may or may not be case sensitive depending on the operating system used.

How to Learn SQL

- https://www.webucator.com/tutorial/learn-sql/simple-selects/introduction-the-northwind-database-reading.cfm#tutorial
  - ➢ Uses Microsoft Northwind database incl. in Access
- https://www.w3schools.com/sql/default.asp
  - ➢ More inclusive: offers MySQL, Oracle, and MS Access specifics



You will get a certificate after passing a quiz

GSA

D2D DATA TO DECISIONS

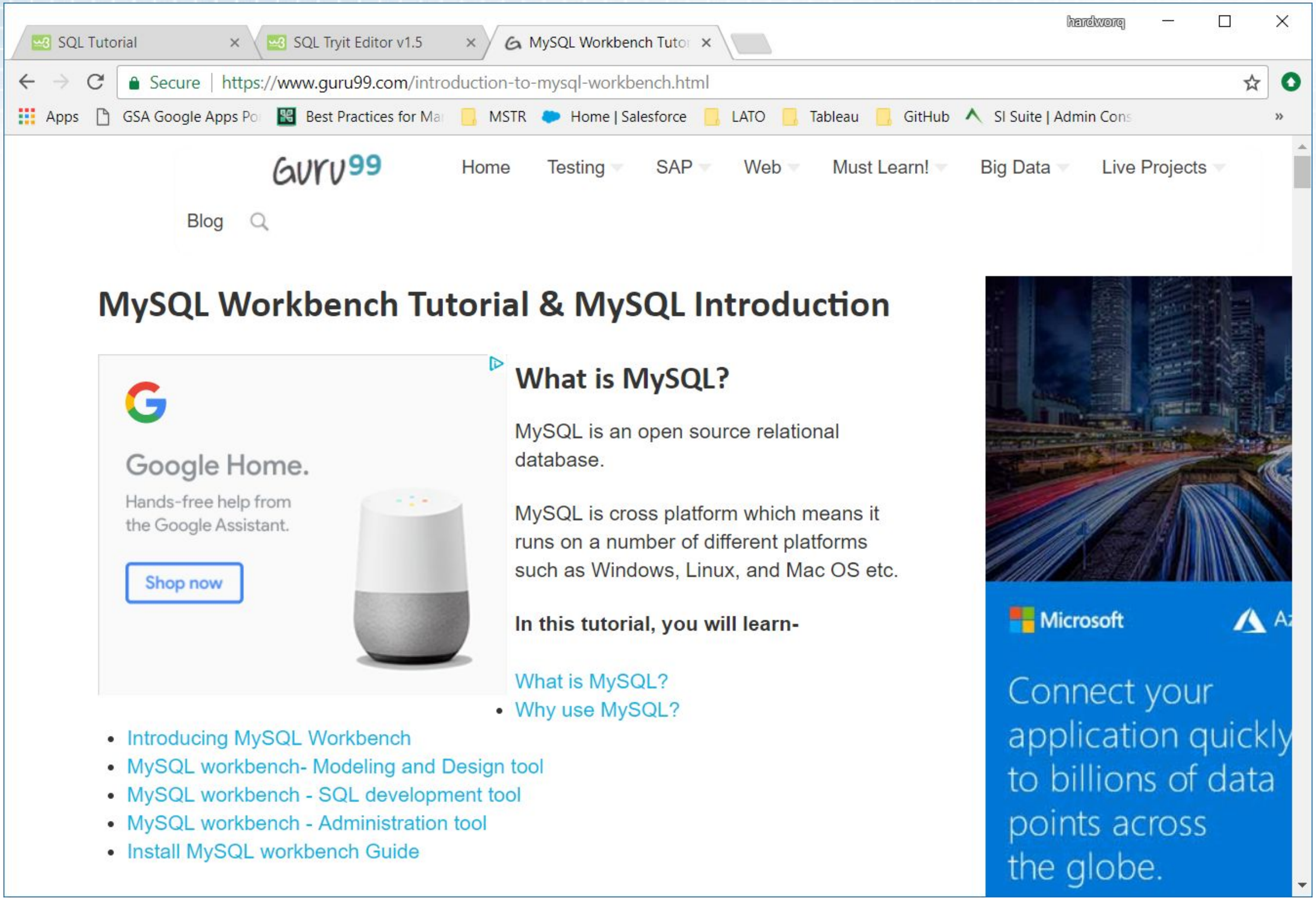
MySQL Workbench

- MySQL is an open source relational database that is cross platform
- MySQL supports multiple storage engines which greatly improve the server performance tuning and flexibility
- MySQL server can be administered using a number of server access mysql tools which include both commercial and open source products:
  - phpMyAdmin - cross platform web based open source server access tool
  - SQLYog - targeted at the windows platform, desktop commercial server access tool
  - MySQL workbench - cross platform open source server access tool.
- MySQL workbench is an integrated development environment for MySQL server
- It has utilities for database modeling and designing, SQL development and server administration
- MySQL workbench is included in DSVD
- https://www.guru99.com/introduction-to-mysql-workbench.html

GSA

D2D
DATA TO DECISIONS

MySQL Workbench Tutorial



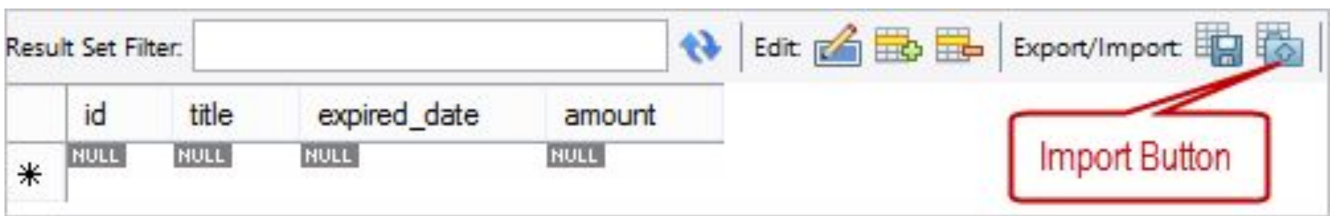
MySQL Workbench Desktop

Import CSV with MySQL Workbench

Step 1: Create table

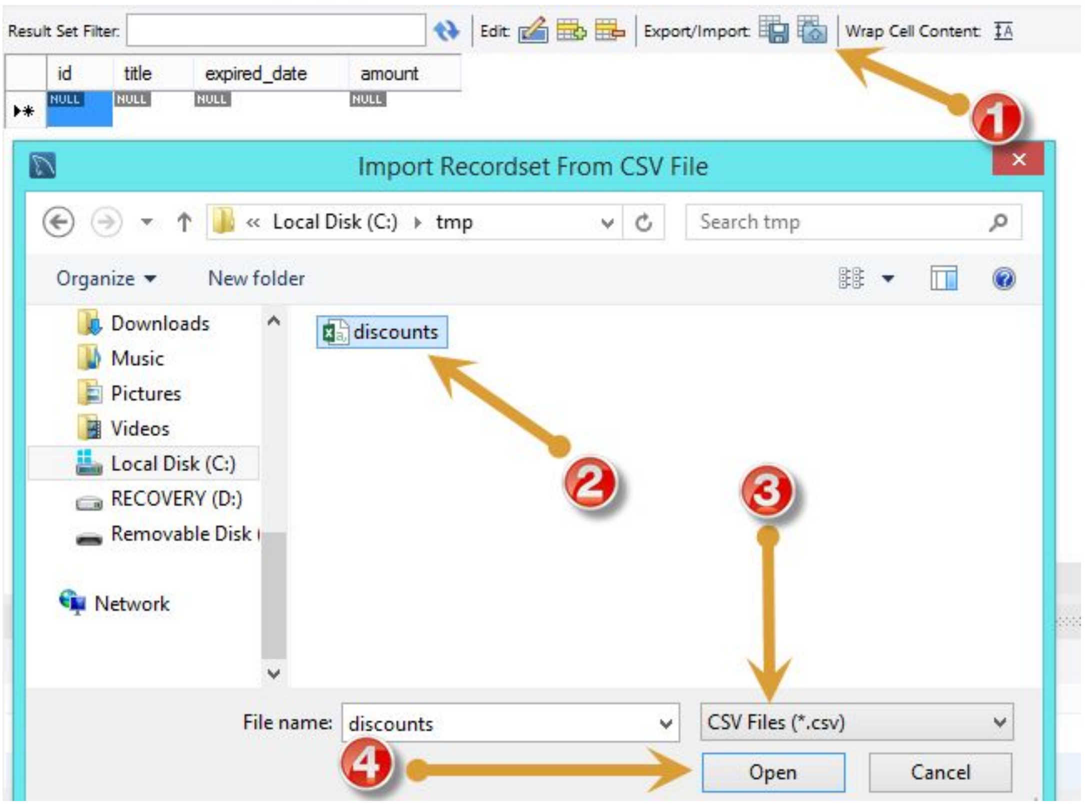
```
CREATE TABLE discounts (
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    expired_date DATE NOT NULL,
    amount DECIMAL(10 , 2 ) NULL,
    PRIMARY KEY (id)
);
```

Step 2: Open the table and click on Import Button



Import CSV with MySQL Workbench (Cont. 1)

Step 3: Follow the dialog

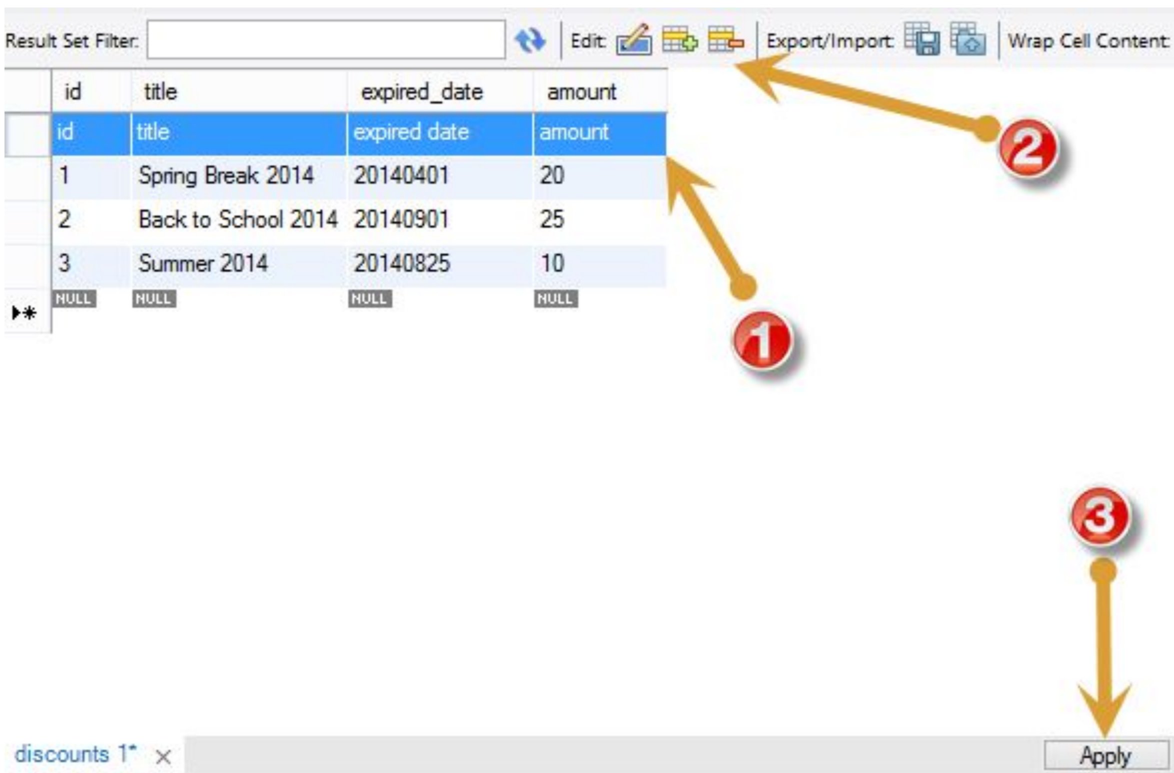


GSA

D2D
DATA TO DECISIONS

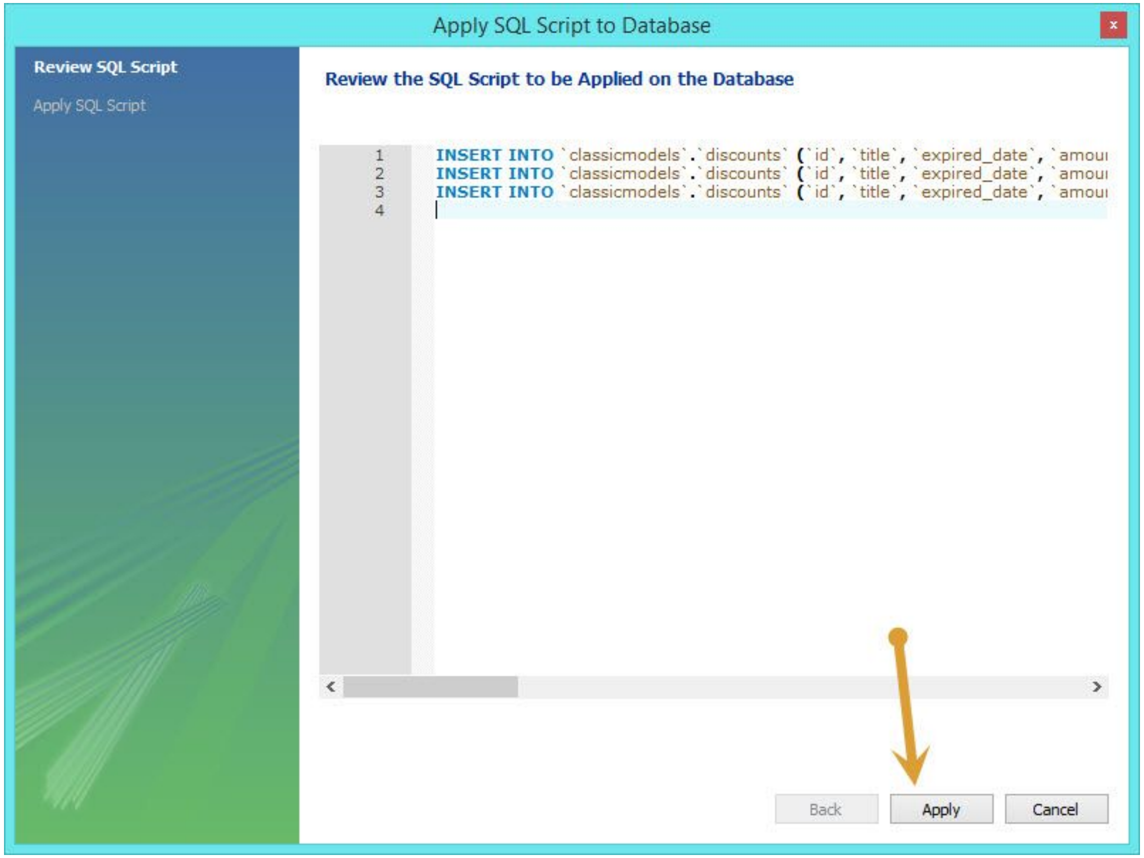
Import CSV with MySQL Workbench (Cont. 2)

Step 4: Review the data



Import CSV with MySQL Workbench (Cont. 3)

Step 5: Upload the data



OLTP Vs. OLAP

- **What is a prime use of your database?**
  - **Operational – OLTP, or**
  - **Analytical – OLAP**
- **OLTP (On-line Transaction Processing)**
  - Large number of short on-line transactions (INSERT, UPDATE, DELETE)
  - The main emphasis for OLTP systems is put on very fast query processing, maintaining data integrity in multi-access environments and an effectiveness measured by number of transactions per second.
  - OLTP database is used to store transactional databases is the entity model
- **OLAP (On-line Analytical Processing)**
  - Relatively low volume of transactions
  - Queries are often very complex and involve aggregations
  - For OLAP systems fast response time is desired
  - OLAP applications are widely used by Data Mining techniques.
  - In OLAP database there is aggregated, historical data, stored in multi-dimensional schemas (usually star schema)

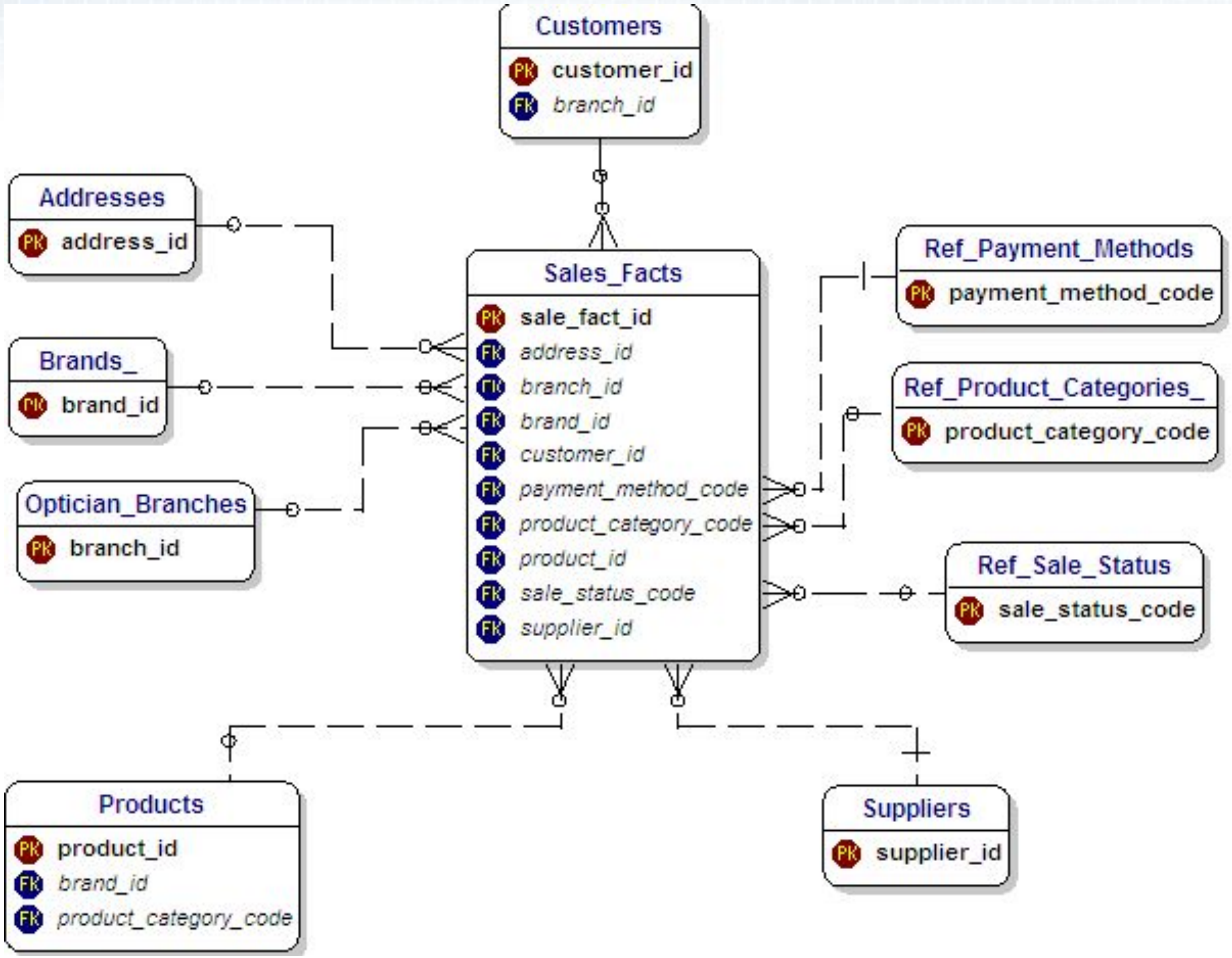
GSA

D2D
DATA TO DECISIONS

Star Schema

- Star schema is the simplest style of data mart schema
- The most widely approach used to develop data warehouses and dimensional data marts
- The star schema consists of one or more fact tables referencing any number of dimension tables
- The star schema gets its name from the dimension tables surrounding it representing the star's points
- Fact table contains
  - Numeric data, and
  - Foreign keys to dimension tables
- Dimension tables contain context / descriptive info for the numeric data

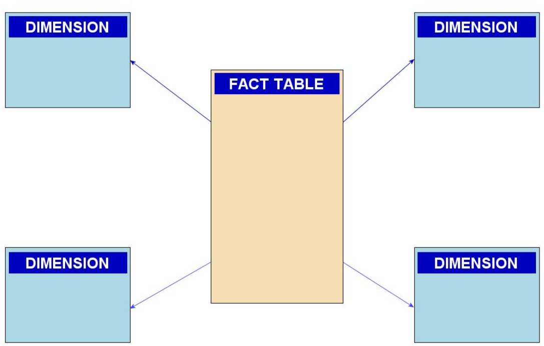
Star Schema Diagram



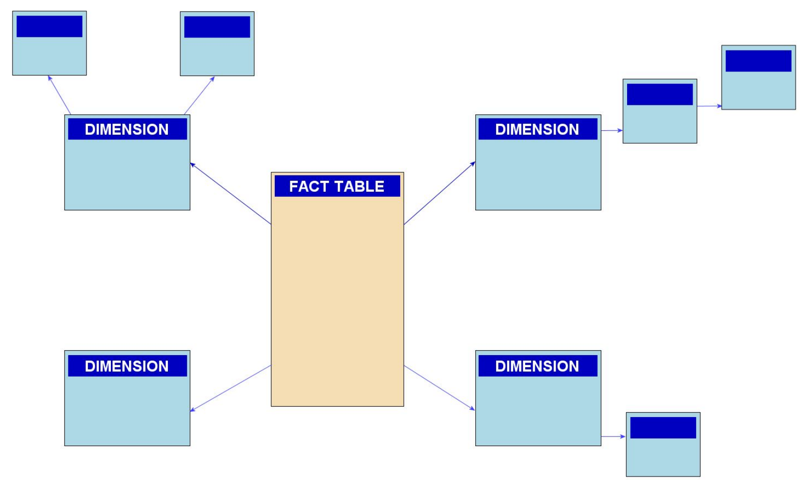
Snowflake Schema

- The snowflake schema is expansion of the star schema
- In the snowflake schema, dimensions are normalized into multiple related tables, whereas the star schema's dimensions are de-normalized with each dimension represented by a single table

Star Schema



Snowflake Schema



GSA

D2D
DATA TO DECISIONS

Snowflake Vs. Star

- Snowflake is highly normalized
- Snowflake better enforces data integrity then star schema
- Requires less space
- The primary disadvantage of the snowflake schema is that the additional levels of attribute normalization adds complexity to source query joins
- Snowflake schemas, in contrast to flat single table dimensions, have been heavily criticized
- The goal of an efficient and compact storage of normalized data comes at the significant cost of poor performance when browsing the joins requires down highly normalized dimension

Review Slide 11

GSA

D2D
DATA TO DECISIONS

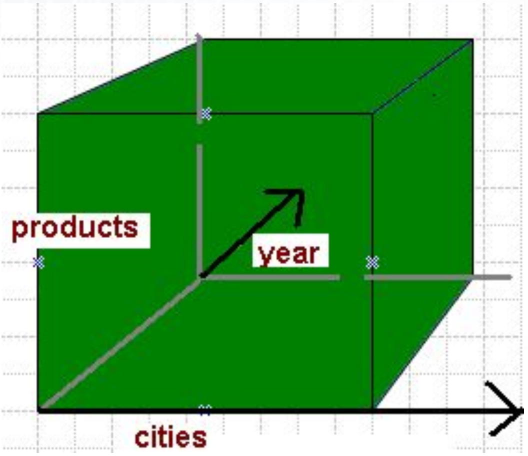
Database view

- A result set of a stored query
- Takes little space to store (query only)
- Hides complexity of the data
- Materialized (pre-executed) view provides high performance
- View has following advantages over tables
  - Can represent a subset of the data
  - Can join multiple table into single virtual table
  - Can aggregate (sum, average, etc.) data
  - Can support drilling

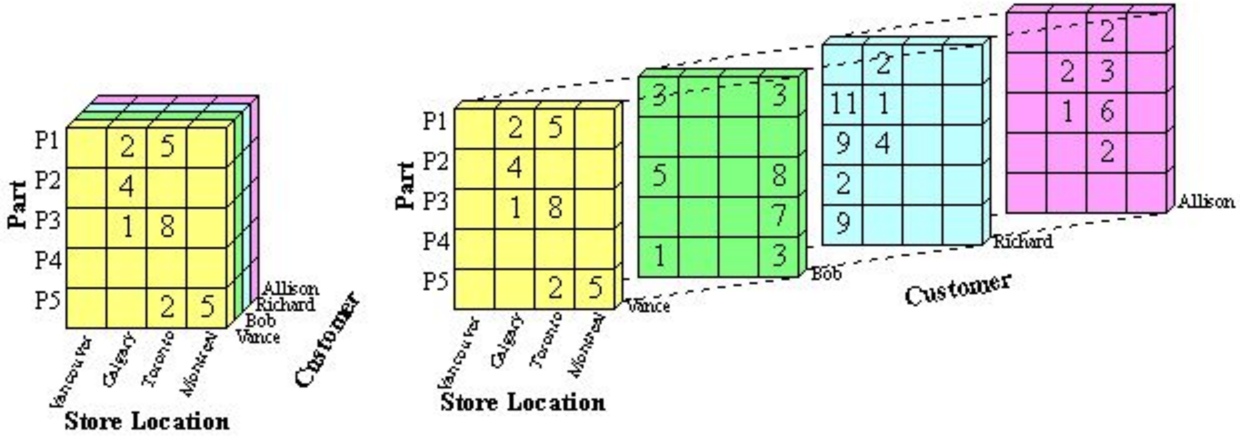
GSA

D2D
DATA TO DECISIONS

OLAP Cubes



- A multi-dimensional array (3-dimensions shown)
- An extension of a spreadsheet / table
- Used to represent data along some measure of interest
- An extension of SQL to allow navigation along a dimension



Pandas Vs SQL

https://pandas.pydata.org/pandas-docs/version/0.22.0/comparison_with_sql.html

- SQL manipulates data in a database
  - You can write stored procedures in SQL
- Pandas is not a data store, it is an in-memory data storage tool
  - This makes Pandas fast, but the data does not persist
  - (You can use Pandas to access database w/ SQL, but that would equal to using SQL)
- Pandas is better with complex analysis
- SQL is better suited for joins
- Recommendation
  - Use SQL to extract and upload data
  - Use Pandas to tidy the data

GSA

D2D
DATA TO DECISIONS

Tidy Data

- **Tidy data principles**
  - ➢ Each variable forms a column
  - ➢ Each observation (tuple) forms a row
  - ➢ Each type of observational unit (dimension) forms a table
- **Five most common problems with messy datasets**
  - ➢ Column headers are values, not variable names.
  - ➢ Multiple variables are stored in one column.
  - ➢ Variables are stored in both rows and columns.
  - ➢ Multiple types of observational units are stored in the same table.
  - ➢ A single observational unit is stored in multiple tables.

Tidying data in R and Python tutorial
https://www.superdatascience.com/wrangling-data-r-python/

- Wrangling the same data set with R and then Python
- Instructions how to
  - Load data
  - Bind rows
  - Select / rename columns
  - Change data types
  - Create data frames
  - Remove duplicates
  - Group by and summarize
  - Join data frames
  - Visualize

Example of Messy Data

Pew data: relationship between income and religion

1: Columns are not names of the variable

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Don’t know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovah’s Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

2: Income categories turned into column names resulting in loosing info: cannot compute average income per religion

Tidied Pew Data

religion	income	noOfPeople
Agnostic	<$10k	27
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$75-100k	122
Agnostic	$100-150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

Question: what else can we do?

Review E. F. Codd principles on slide 10

GSA

D2D DATA TO DECISIONS

pandas.melt

pandas.melt(frame, id_vars=None, value_vars=None, var_name=None, value_name='value', col_level=None)

- "Unpivots" a DataFrame from wide format to long format, optionally leaving id variables set
- Parameters :
  - frame : DataFrame
  - id_vars : tuple, list, or ndarray
  - value_vars : tuple, list, or ndarray
  - var_name : scalar, if None uses frame.column.name or 'variable'
  - value_name : scalar, default 'value'
  - col_level : scalar, if columns are a MultiIndex then use this level to melt

GSA

D2D
DATA TO DECISIONS

Tidying Pew Data with Pandas

```
pew_raw = pd.read_csv('D:\\Acuity\\D2D\\Data Science Training\\Python\\pew_raw.csv')

pew_long = pd.melt(pew_raw,
                   ["religion"],
                   var_name="income",
                   value_name="count")
pew_long = pew_long.sort_values(by=["religion"])
pew_long.head(10)
```

Microsoft Excel
a Separated Va

```
pew_avg_cat = pd.read_csv('D:\\Acuity\\D2D\\Data Science Training\\Python\\pew_avg_cat.csv')

pew_long_avg = pd.melt(pew_avg_cat,
                       ["religion"],
                       var_name="avg.income",
                       value_name="count")
pew_long_avg = pew_long_avg.sort_values(by=["religion"])
pew_long_avg.head(10)
```

Microsoft Excel
a Separated Va

GSA

D2D
DATA TO DECISIONS

Reshape by Pivoting

- Data is often stored in CSV files or databases in so-called “stacked” or “record” format

	date	variable	value
0	2000-01-03	A	0.469112
1	2000-01-04	A	-0.282863
2	2000-01-05	A	-1.509059
3	2000-01-03	B	-1.135632
4	2000-01-04	B	1.212112

- However, a preferred format for time-series analysis is

variable date	A	B	C	D
2000-01-03	0.469112	-1.135632	0.119209	-2.104569
2000-01-04	-0.282863	1.212112	-1.044236	-0.494929
2000-01-05	-1.509059	-0.173215	-0.861849	1.071804

- To reshape the data into this form, we use the DataFrame.pivot() method
- https://pandas.pydata.org/pandas-docs/stable/reshaping.html

Reshape by pivoting in Pandas

```
# Reshape by pivoting

DateVarValRaw = pd.read_csv('D:\\Acuity\\D2D\\Data Science Training\\Python\\DataVariableValue.csv')

DateVarValPiv = DateVarValRaw.pivot(index='date', columns='variable', values='value')
```

Microsoft Excel
ıa Separated Val

Pandas multi-level Indexing

```
tips = sns.load_dataset('tips')

# can aggregate
tips_smoker = tips.groupby('smoker').mean()
tips_smoker

tips_smoker.index
tips_smoker.reset_index()

tips_smoker_time =
tips.groupby(['smoker','time']).mean()
tips_smoker_time
tips_smoker_time.index

tips_smoker_time.swaplevel()
tips_smoker_time.unstack()
tips_smoker_time.unstack(level = 0)
```

```
tips_size = tips.groupby(['size', 'time']).mean()
tips_size

tips_count = tips.groupby(['size', 'time']).size()
tips_count

tips_size_smoker = tips.groupby(['size', 'smoker']).size()
tips_size_smoker
# explain what is size

tips_smoker_time.swaplevel()
tips_smoker_time.unstack()
tips_smoker_time.unstack(level = 0)
```

https://pandas.pydata.org/pandas-docs/stable/advanced.html

http://shop.oreilly.com/product/0636920034919.do

Python Data Science Handbook

Essential Tools for Working with Data

By Jake VanderPlas

Publisher: O'Reilly Media
Release Date: November 2016
Pages: 541

For many researchers, Python is a first-class tool mainly because of its libraries for storing, manipulating, and gaining insight from data. Several resources exist for individual pieces of this data science stack, but only with the Python Data Science Handbook do you get them all—IPython, NumPy, Pandas, Matplotlib, Scikit-Learn, and other related tools.

- https://jakevdp.github.io/PythonDataScienceHandbook/
  - https://jakevdp.github.io/PythonDataScienceHandbook/03.10-working-with-strings.html
  - https://jakevdp.github.io/PythonDataScienceHandbook/03.09-pivot-tables.html
  - https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html

GSA

D2D
DATA TO DECISIONS

http://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Data Wrangling with pandas Cheat Sheet
http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

F M A

Each **variable** is saved in its own **column**

&

F M A

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
     [[4, 7, 10],
      [5, 8, 11],
      [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```
Specify values for each row.

n	v	a	b	c
d	1	4	7	10
d	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
           names=['n','v'])))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
     )
```

Reshaping Data – Change the layout of a data set

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length','Height'])`
Drop columns from DataFrame

Subset Observations (Rows)

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)

`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

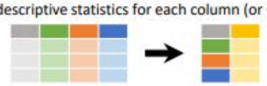
GSA

D2D DATA TO DECISIONS

http://pandas.pydata.org/Pandas_Cheat_Sheet.pdf (Cont.)

Summarize Data

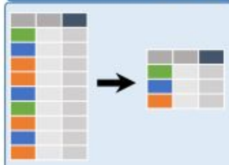
`df['w'].value_counts()`
Count number of rows with each unique value of variable
`len(df)`
# of rows in DataFrame.
`df['w'].nunique()`
# of distinct values in a column.
`df.describe()`
Basic descriptive statistics for each column (or GroupBy)



pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.
`count()`
Count non-NA/null values of each object.
`median()`
Median value of each object.
`quantile([0.25,0.75])`
Quantiles of each object.
`apply(function)`
Apply function to each object.
`min()`
Minimum value in each object.
`max()`
Maximum value in each object.
`mean()`
Mean value of each object.
`var()`
Variance of each object.
`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:
`size()`
Size of each group.
`agg(function)`
Aggregate group using function.

Windows

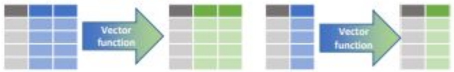
`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.
`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.
`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.
`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

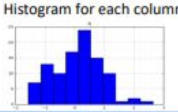
`max(axis=1)`
Element-wise max.
`clip(lower=-10,upper=10)`
Trim values at input thresholds
`min(axis=1)`
Element-wise min.
`abs()`
Absolute value.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

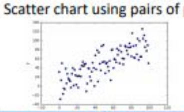
`shift(1)`
Copy with values shifted by 1.
`rank(method='dense')`
Ranks with no gaps.
`rank(method='min')`
Ranks. Ties get min rank.
`rank(pct=True)`
Ranks rescaled to interval [0, 1].
`rank(method='first')`
Ranks. Ties go to first value.
`shift(-1)`
Copy with values lagged by 1.
`cumsum()`
Cumulative sum.
`cummax()`
Cumulative max.
`cummin()`
Cumulative min.
`cumprod()`
Cumulative product.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

bdf

x1	x3
A	T
B	F
D	T

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

zdf

x1	x2
B	2
C	3
D	4

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

Q & A