W4111 – Introduction to Databases
Section 002, Spring 2024

Lecture 7: ER, Relational, SQL (VI)

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

W4111 – Introduction to Databases
Section 002, Spring 2024

Lecture 7: ER, Relational, SQL (VI)

We will start in a couple of minutes.

Contents

Contents

- WITH clause and Common Table Expressions revisited.

SQL

WITH Clause
Common Table Expressions

With Clause

- The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs.
- Find all departments with the maximum budget

```
with max_budget (value) as
        (select max(budget)
         from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

Common Table Expressions

- CTE: (https://www.essentialsql.com/introduction-common-table-expressions-ctes/)
  - "The Common Table Expressions or CTE's for short are used within SQL (...) to simplify complex joins and subqueries, ..."
  - "A CTE (Common Table Expression) defines a temporary result set which you can then use in a SELECT statement. It becomes a convenient way to manage complicated queries."
  - There are two types of CTEs:
    - Non-recursive.
    - Recursive (note, recursive SQL weirds me out).
  - The benefits are clarity and improved quality through incremental development.

- Basic syntax by example.
  - There may be several CTEs.
  - A CTE can reference other CTEs.

```
With Employee_CTE (EmployeeNumber, Title)
AS
(
SELECT NationalIDNumber,
       JobTitle
FROM   HumanResources.Employee
)
SELECT EmployeeNumber,
       Title
FROM   Employee_CTE
```

CTE (Common Table Expression)

Query Using CTE

Common Table Expressions – Example

```
WITH career_batting (playerid, h, ab, bb, hr, rbi) AS
  (
    SELECT playerid, sum(h) AS h, sum(ab) AS ab, sum(bb) AS bb,
        sum(hr) AS hr, sum(rbi) AS rbi
      FROM batting GROUP BY playerid
  ),
  career_pitching (playerid, w, l, ipouts, er) AS
  (
    SELECT playerid, sum(w) AS w, sum(l) AS l, sum(ipouts) as ipouts, sum(er) AS er
      FROM pitching GROUP BY playerid
  ),
  career_summary (playerid, h, ab, bb, hr, rbi, bavg, obp, w, l, ipouts, er, era) AS
  (
    SELECT career_batting.playerid, h, ab, bb, hr, rbi,
      IF(ab<500, NULL, round(h/ab, 3)) AS bavg,
      IF(ab<500, NULL, round((h+bb)/(ab + bb), 3)) AS obp,
      w, l, ipouts, er, round((er/(ipouts/3))*9, 3) AS era
    FROM
      career_batting join career_pitching using(playerid)
  )
SELECT
  playerid, nameLast, nameFirst, career_summary.*
FROM
  people JOIN career_summary using(playerid);
```

- Producing a career summary requires several tasks:
  1. Computing batting totals
  2. Computing pitching totals
  3. Applying formulas to produce derived averages and metrics.
  4. Joining (1), (2) and (3) into a career summary.
  5. Joining with people to bring in personal information.
- Producing the table is possible with a single SELECT statement, but ... ...
  - Developing incrementally one simpler query at a time is easier.
  - The resulting query is easier to understand and maintain.

Common Table Expressions – Example

```
WITH career_batting (playerid, h, ab, bb, hr, rbi) AS
  (
    SELECT playerid, sum(h) AS h, sum(ab) AS ab, sum(bb) AS bb,
        sum(hr) AS hr, sum(rbi) AS rbi
      FROM batting GROUP BY playerid
  ),
  career_p
  (
    SELE
      FR
  ),
  career_
  (
    SELE
      IF(
      IF(a
      w, l, ipo
    FROM
      career_batting join career_pitching using(playerid)
  )
SELECT
  playerid, nameLast, nameFirst, career_summary.*
FROM
  people JOIN career_summary using(playerid);
```

- I find it difficult to impossible to write complex queries without using CTEs.
- I think about incremental steps to produce the result.
- Write and test the CTE by adding one step at a time.
- Please, please, please:
  - Use CTEs.
  - Use DataGrip

- Producing a career summary requires several tasks:
  1. Computing batting totals
  2. ls
  3. duce
  4. trics.
  5. to
  6. ing in
  7. ssible with a

single SELECT statement, but ... ...

- Developing incrementally one simpler query at a time is easier.
- The resulting query is easier to understand and maintain.

Recursive Queries

Recursion in SQL

- SQL:1999 permits recursive view definition
- Example: find which courses are a prerequisite, whether directly or indirectly, for a specific course

```
with recursive rec_prereq(course_id, prereq_id) as (
        select course_id, prereq_id
        from prereq
    union
        select rec_prereq.course_id, prereq.prereq_id,
        from rec_rereq, prereq
        where rec_prereq.prereq_id = prereq.course_id
    )
select ∗
from rec_prereq;
```

This example view, *rec_prereq,* is called the *transitive closure* of the *prereq* relation

The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.
  - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *prereq* with itself
    - This can give only a fixed number of levels of managers
    - Given a fixed non-recursive query, we can construct a database with a greater number of levels of prerequisites on which the query will not work
    - Alternative: write a procedure to iterate as many times as required
      - See procedure *findAllPrereqs* in book

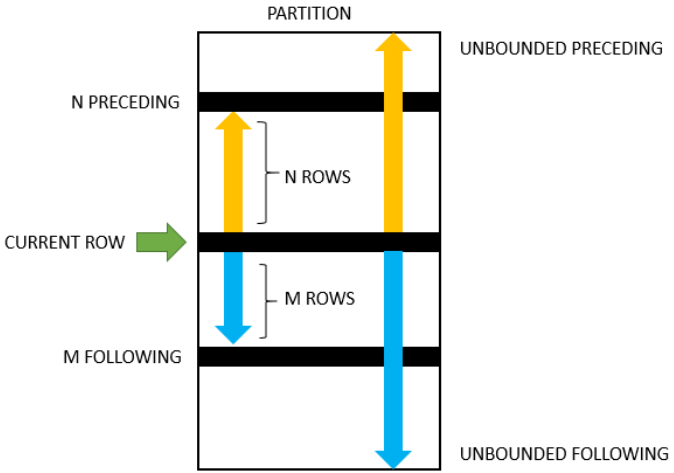
- Since SQL is not Turing complete, there are many computations it cannot perform.
- Recursion enables some scenarios, but recursion in general is hard to understand.
- Switch to notebook

Advanced Aggregation Window Functions

Advanced Aggregates and Window Functions

- The aggregation functions in SQL are powerful.
- There are some scenarios that are very difficult. SQL and databases add support for more advanced capabilities:
  - Ranking
  - Windows
  - Pivot, Slide and Dice, but we will cover these with a different technology (OLAP) later in the semester.

```
<window function name>( )   OVER (
        PARTITION BY <expression>
        ORDER BY <expression> [ASC | DESC]
    )
```



  - This is powerful, a little complex and requires trial and error.

Switch to Notebook

- Note – This takes a lot of practice and “help.”
- To verify, in some cases you can
  - Use a subset of of the data, e.g. filter by country.
  - Compute the aggregate using normal aggregation.
  - Copy into Excel/Sheets, write some formulas and verify.
- This is quite powerful. There will be a couple of take home HW/exam questions, but this takes tinkering.

Security

Security Concepts (Terms from Wikipedia)

- Definitions:
  - "A (digital) identity is information on an entity used by computer systems to represent an external agent. That agent may be a person, organization, application, or device."
  - "Authentication is the act of proving an assertion, such as the identity of a computer system user. In contrast with identification, the act of indicating a person or thing's identity, authentication is the process of verifying that identity."
  - "Authorization is the function of specifying access rights/privileges to resources, ... More formally, "to authorize" is to define an access policy. ... During operation, the system uses the access control rules to decide whether access requests from (authenticated) consumers shall be approved (granted) or disapproved.
  - "Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. Members or staff (or other system users) are assigned particular roles, and through those role assignments acquire the permissions needed to perform particular system functions."
  - "In computing, privilege is defined as the delegation of authority to perform security-relevant functions on a computer system. A privilege allows a user to perform an action with security consequences. Examples of various privileges include the ability to create a new user, install software, or change kernel functions."

- SQL and relational database management systems implementing security by:
  - Creating identities and authentication policies.
  - Creating roles and assigning identities to roles.
  - Granting and revoking privileges to/from roles and identities.

Authorization

- We may assign a user several forms of authorizations on parts of the database.
  - **Read** - allows reading, but not modification of data.
  - **Insert** - allows insertion of new data, but not modification of existing data.
  - **Update** - allows modification, but not deletion of data.
  - **Delete** - allows deletion of data.
- Each of these types of authorizations is called a **privilege**. We may authorize the user all, none, or a combination of these types of privileges on specified parts of a database, such as a relation or a view.

Authorization (Cont.)

- Forms of authorization to modify the database schema
  - **Index** - allows creation and deletion of indices.
  - **Resources** - allows creation of new relations.
  - **Alteration** - allows addition or deletion of attributes in a relation.
  - **Drop** - allows deletion of relations.

Authorization Specification in SQL

- The **grant** statement is used to confer authorization

  grant <privilege list> **on** <relation or view > **to** <user list>
- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Example:
  - **grant select on** *department* **to** Amit, Satoshi
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- **select**: allows read access to relation, or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

    grant select on *instructor* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

  revoke <privilege list> **on** <relation or view> **from** <user list>

- Example:

  revoke select on *student* **from** U_1, U_2, U_3

- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Roles

- A **role** is a way to distinguish among various users as far as what these users can access/update in the database.
- To create a role we use:

  create a role <name>
- Example:
  - **create role** instructor
- Once a role is created we can assign “users” to the role using:
  - **grant** <role> **to** <users>

Roles Example

- **create role** instructor;
- **grant** *instructor* **to** Amit**;**
- Privileges can be granted to roles:
  - **grant select on** *takes* **to** *instructor*;
- Roles can be granted to users, as well as to other roles
  - **create role** *teaching_assistant*
  - **grant** *teaching_assistant* **to** *instructor*;
    - *Instructor* inherits all privileges of *teaching_assistant*
- Chain of roles
  - **create role** *dean*;
  - **grant** *instructor* **to** *dean*;
  - **grant** *dean* **to** Satoshi;

Authorization on Views

- **create view** *geo_instructor* **as**
  (**select** *
  from *instructor*
  where *dept_name* = 'Geology');
- **grant select on** *geo_instructor* **to** *geo_staff*
- Suppose that a *geo_staff* member issues
  - **select** *
    from *geo_instructor*;
- What if
  - *geo_staff* does not have permissions on *instructor?*
  - Creator of view did not have some permissions on *instructor?*

Other Authorization Features

- **references** privilege to create foreign key
  - **grant reference** (*dept_name*) **on** *department* **to** Mariano;
  - Why is this required?
- transfer of privileges
  - **grant select on** *department* **to** Amit **with grant option**;
  - **revoke select on** *department* **from** Amit, Satoshi **cascade**;
  - **revoke select on** *department* **from** Amit, Satoshi **restrict**;
  - And more!

Note:

- Like in many other cases, SQL DBMS have product specific variations.

Switch to notebook.

Worked Example –
Or
Mock Prof. Ferguson While He Codes

Scenario

- The data model is:
  - Students
  - Faculty
  - An abstract base type Person
- We will
  - Draw the logical and a partial physical ER diagram.
  - ~~Generate some test data using Mockaroo. We will load the data and create the initial schema~~.
  - Give you a bunch of data. --> Load → Tables → constraints
  - Implement a three-table solution, including indexes, views and constraints.
  - A function to compute UNIs.
  - Triggers to automatically assign UNIs.
  - Stored procedures to create student, faculty.
  - Create non-root users.
  - Revoke SQL operations to prevent people from making mistakes.
- Snicker at Prof. Ferguson. I put this last, but it will like occur during all steps.

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.

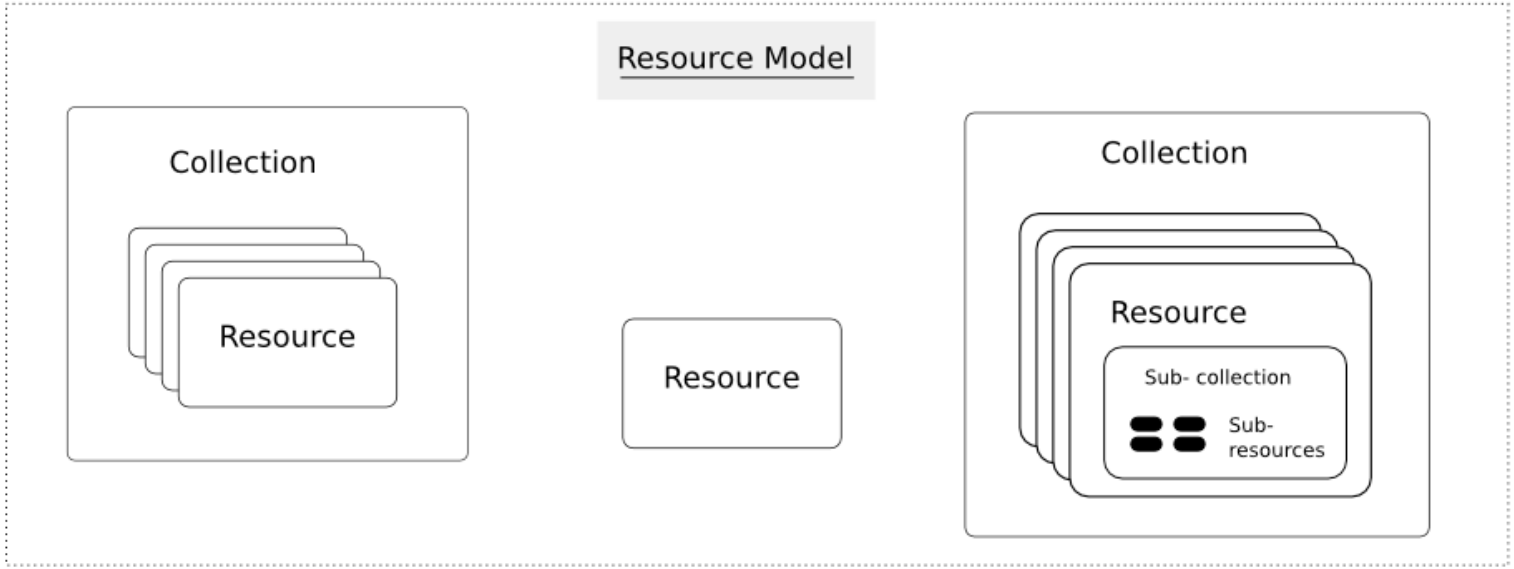
Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP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

Creating RESTFul Webservice

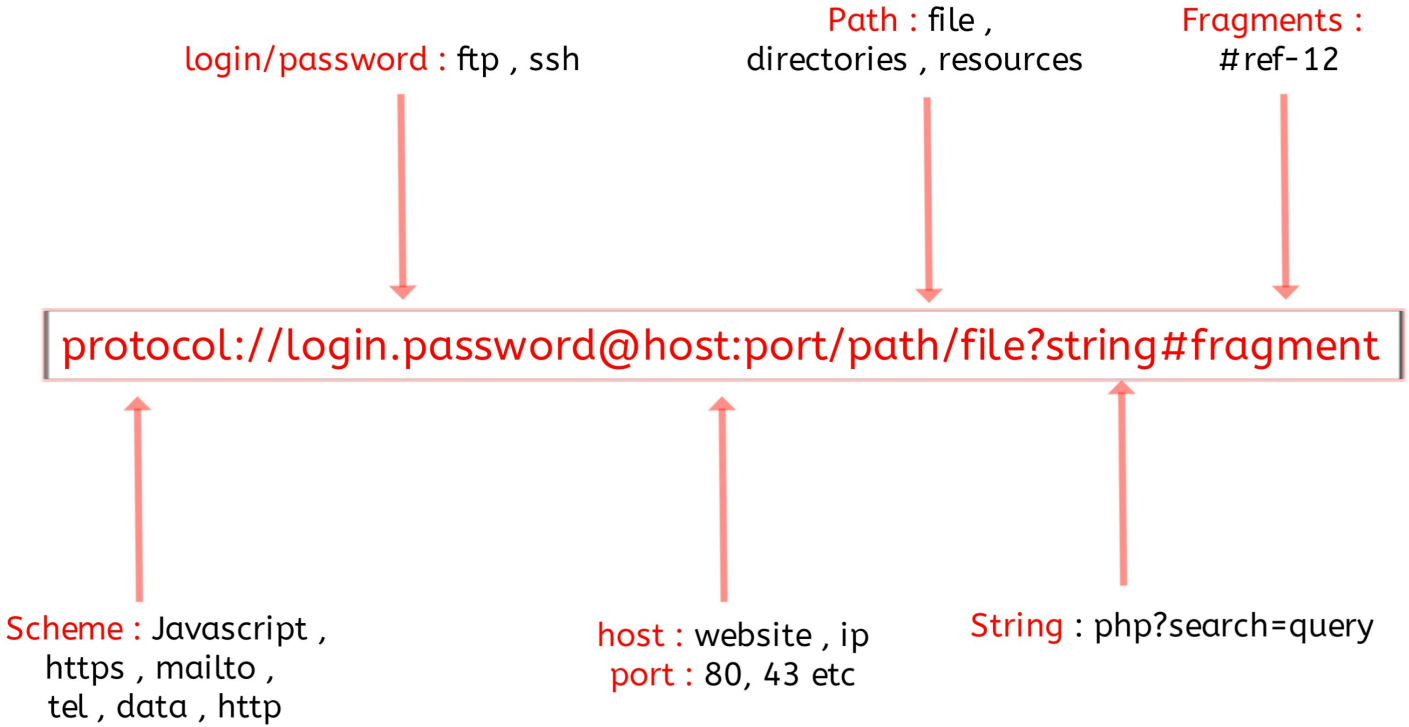
In next chapters, we'll create a webservice say user management with following functionalities –

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



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

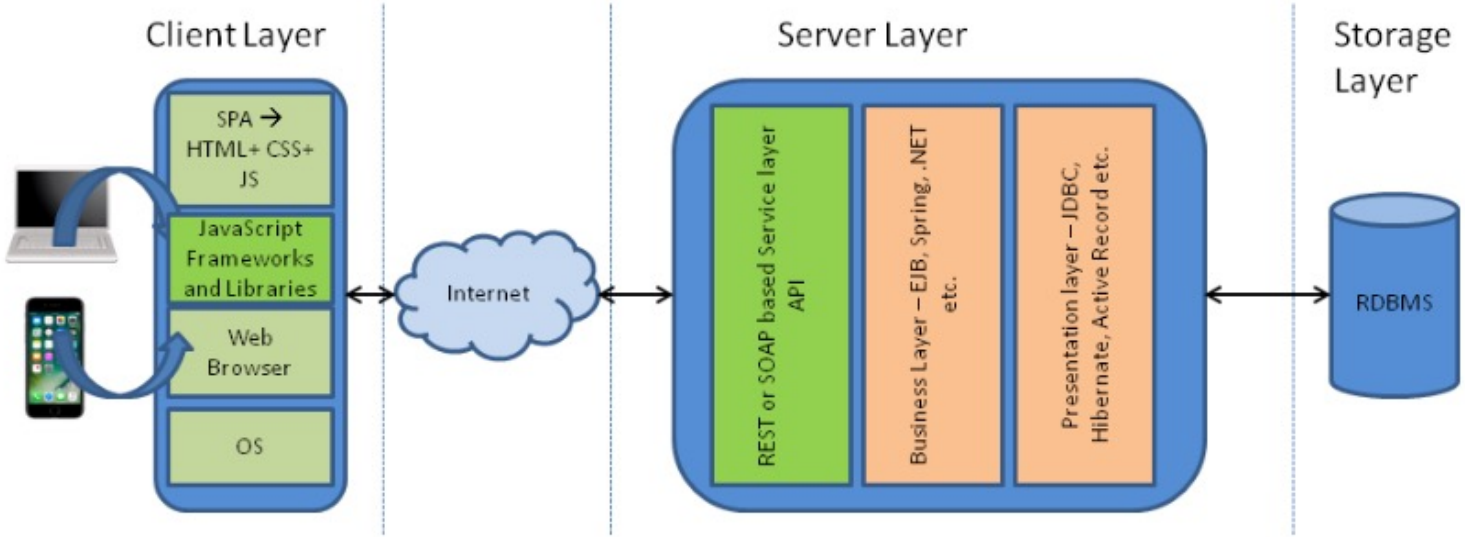


Diagram 2: The moving of the Web Layer from the Server to the Client

Walkthrough Code

- /Users/donaldferguson/Dropbox/000/000-A-Current-Examples/W4111-FastAPI-IMDB-Solution
- /Users/donaldferguson/Dropbox/000/000-A-Current-Examples/current-dashboard

Big Data
Business Intelligence
Decision Support
(Preview to enable project)

Decision Support Systems

- **Decision-support systems** are used to make business decisions, often based on data collected by on-line transaction-processing systems.
- Examples of business decisions:
  - What items to stock?
  - What insurance premium to change?
  - To whom to send advertisements?
- Examples of data used for making decisions
  - Retail sales transaction details
  - Customer profiles (income, age, gender, etc.)

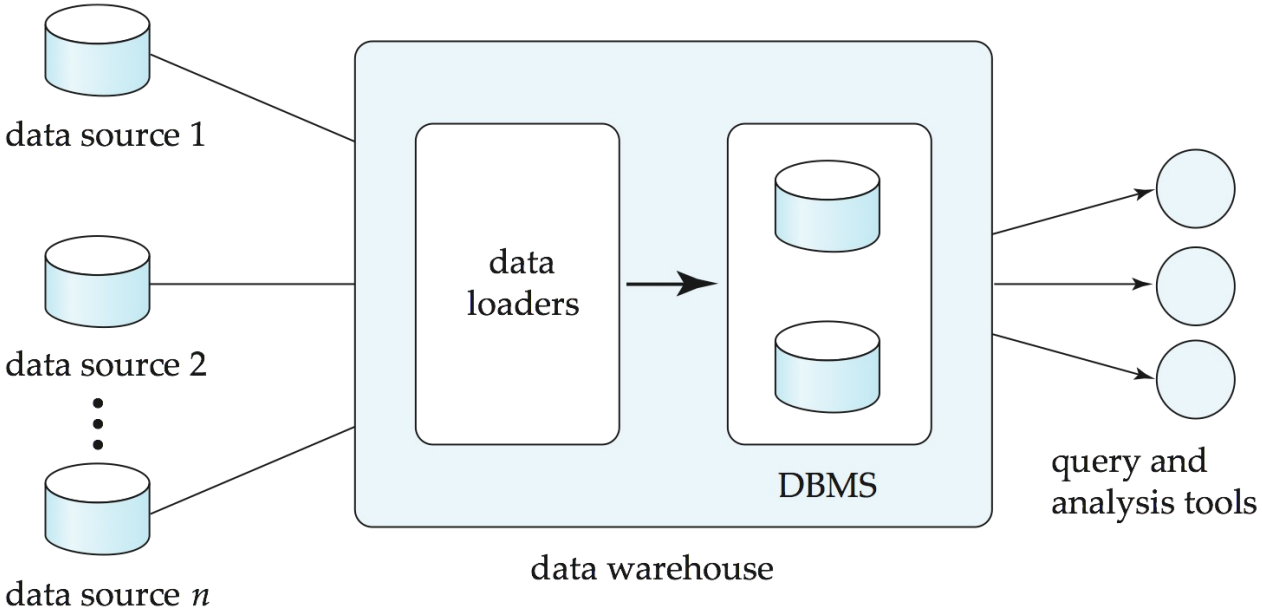
Decision-Support Systems: Overview

- **Data analysis** tasks are simplified by specialized tools and SQL extensions
  - Example tasks
    - For each product category and each region, what were the total sales in the last quarter and how do they compare with the same quarter last year
    - As above, for each product category and each customer category
- **Statistical analysis** packages (e.g., : S++) can be interfaced with databases
  - Statistical analysis is a large field, but not covered here
- **Data mining** seeks to discover knowledge automatically in the form of statistical rules and patterns from large databases.
- A **data warehouse** archives information gathered from multiple sources, and stores it under a unified schema, at a single site.
  - Important for large businesses that generate data from multiple divisions, possibly at multiple sites
  - Data may also be purchased externally

Data Warehousing

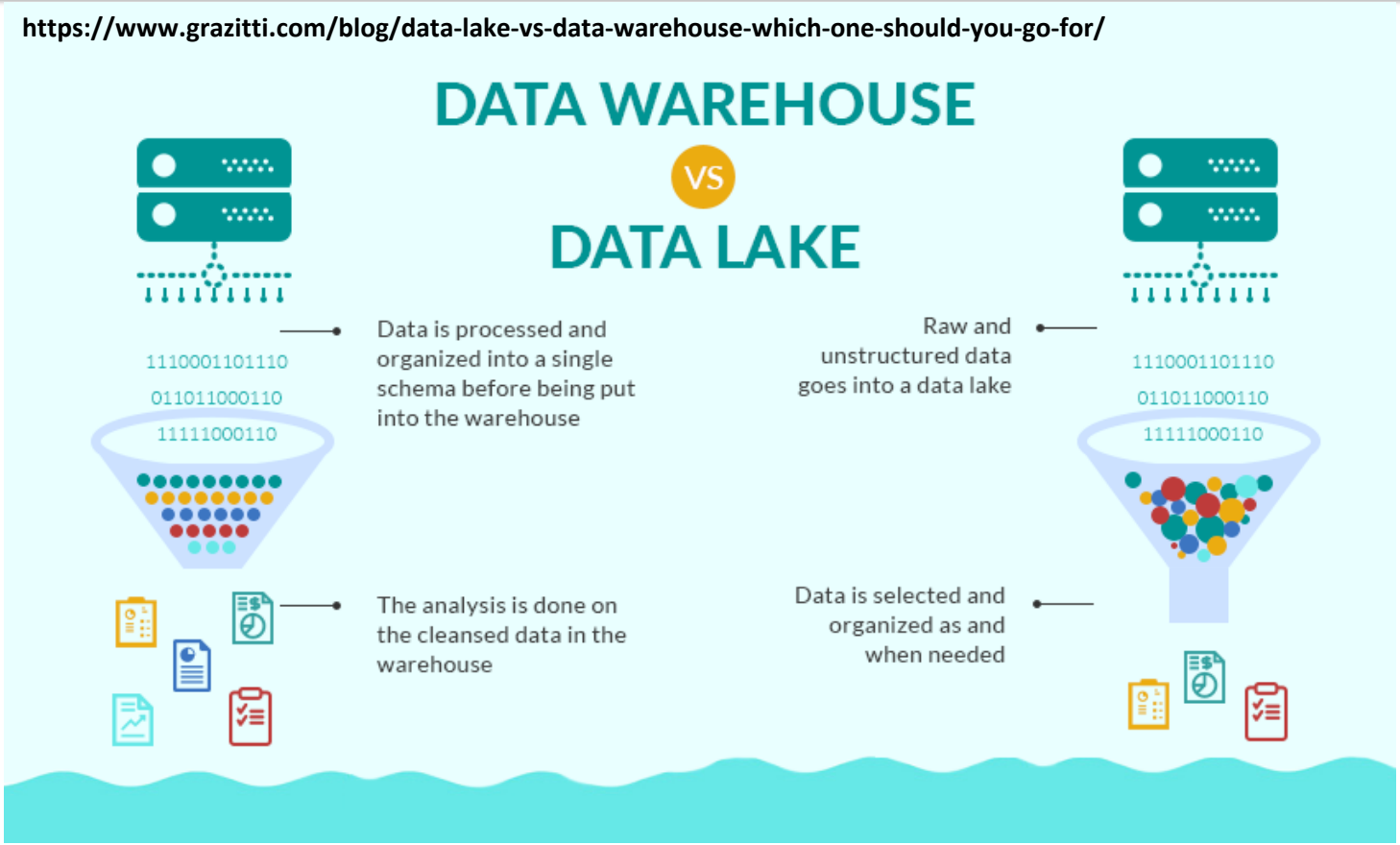
- Data sources often store only current data, not historical data
- Corporate decision making requires a unified view of all organizational data, including historical data
- A **data warehouse** is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site
  - Greatly simplifies querying, permits study of historical trends
  - Shifts decision support query load away from transaction processing systems

Data Warehousing



Data Warehouse vs Data Lake

https://www.grazitti.com/blog/data-lake-vs-data-warehouse-which-one-should-you-go-for/



Simplistic: Data Warehouse vs Data Lake

https://panoply.io/data-warehouse-guide/data-warehouse-vs-data-lake/

DATA WAREHOUSE	vs.	DATA LAKE
structured, processed	DATA	structured / semi-structured / unstructured, raw
schema-on-write	PROCESSING	schema-on-read
expensive for large data volumes	STORAGE	designed for low-cost storage
less agile, fixed configuration	AGILITY	highly agile, configure and reconfigure as needed
mature	SECURITY	maturing
business professionals	USERS	data scientists et. al.

Design Issues

- *When and how to gather data*
  - **Source driven architecture**: data sources transmit new information to warehouse, either continuously or periodically (e.g., at night)
  - **Destination driven architecture**: warehouse periodically requests new information from data sources
  - Keeping warehouse exactly synchronized with data sources (e.g., using two-phase commit) is too expensive
    - Usually OK to have slightly out-of-date data at warehouse
    - Data/updates are periodically downloaded form online transaction processing (OLTP) systems.
- *What schema to use*
  - Schema integration

More Warehouse Design Issues

- *Data cleansing*
  - E.g., correct mistakes in addresses (misspellings, zip code errors)
  - **Merge** address lists from different sources and **purge** duplicates
- *How to propagate updates*
  - Warehouse schema may be a (materialized) view of schema from data sources
- *What data to summarize*
  - Raw data may be too large to store on-line
  - Aggregate values (totals/subtotals) often suffice
  - Queries on raw data can often be transformed by query optimizer to use aggregate values

Sample Projects

Projects

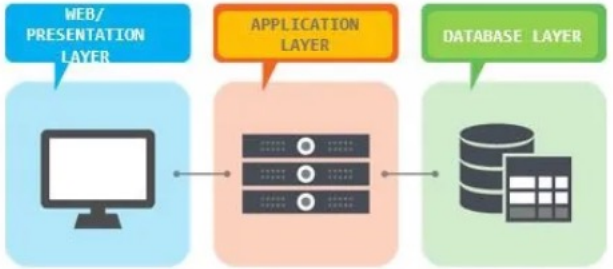
- The programming track will implement a simple, full stack web application.

Full-stack Web Developer

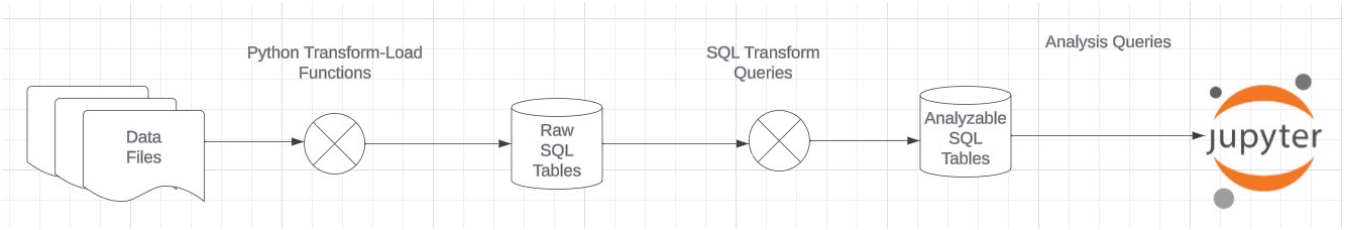
A full-stack web developer is a person who can develop both **client** and **server** software.

In addition to mastering HTML and CSS, he/she also knows how to:

- Program a **browser** (e.g. using JavaScript, jQuery, Angular, or Vue)
- Program a **server** (e.g. using PHP, ASP, Python, or Node)
- Program a **database** (e.g. using SQL, SQLite, or MongoDB)



- The non-programming track will implement a data engineering and visualization process in a Jupyter notebook via a simple data warehouse.



Game of Thrones

- Bottom-Up Data Mapping:
  - "Nouns" usually map to Entity/Entity Set.
  - Nouns inside other nouns often map to:
    - Attribute
    - Relationship
  - Verbs often map to relationships.
  - Adjectives usually map to properties.
- We will start with a subset of the information:
  - Game of Thrones:
    - Episodes
    - Characters
  - IMDB:
    - names_basics
    - title_basics

- Entities Sets
  - Character
  - Season
  - Episode
  - Scene
  - Location, Sublocation
  - … …
- Relationships
  - Character – Scene
  - Character – Character (e.g. KilledBy)
  - Season – IMDB Title
  - Character – IMDB Name
  - … …

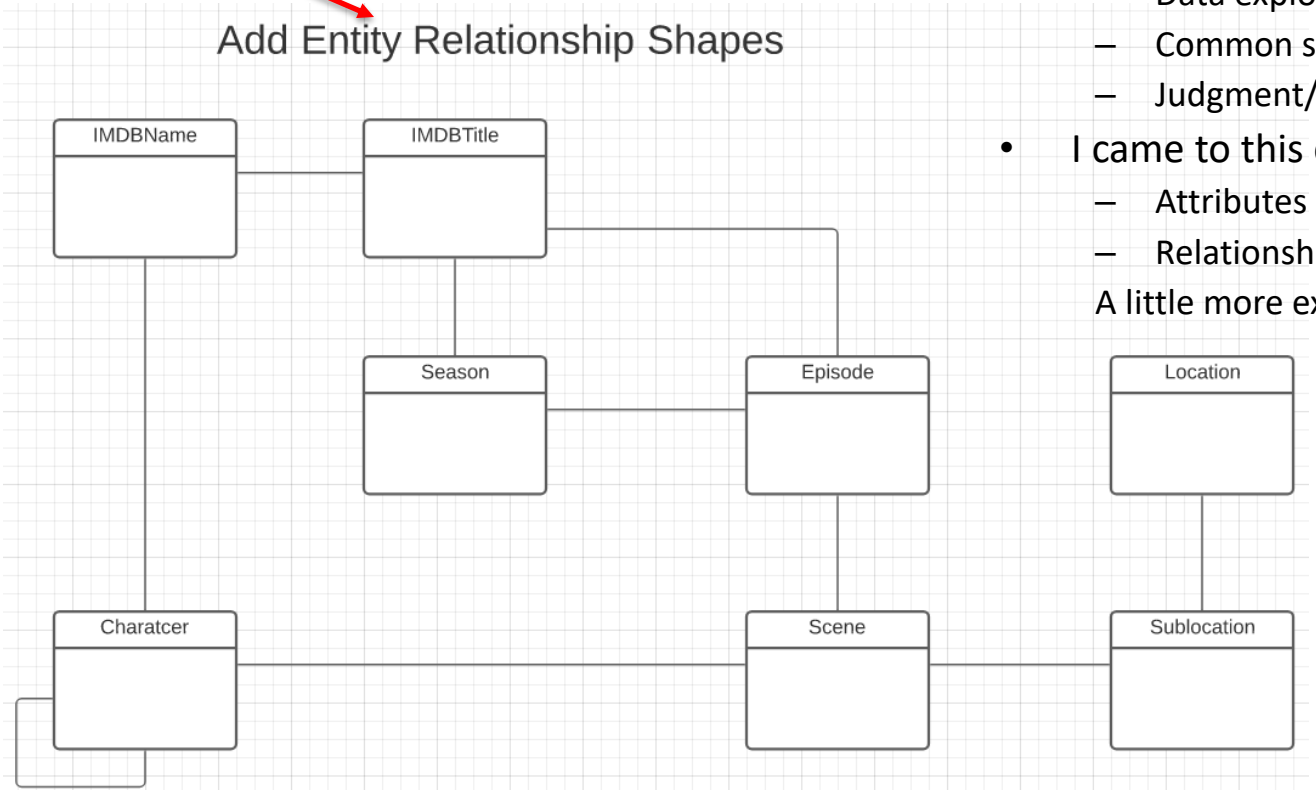
Game of Thrones

- Bottom-Up Data Mapping:
  - –
  - –
  - –
  - –
- We
  info
  - –
  - – IM
    - names_basics
    - title_basics
- Entities Sets
  - – Character – IMDB Name
  - – … …

1. IMDB: https://developer.imdb.com/non-commercial-datasets/
   - Do not download.
   - Despite being “tiny” compared to the reall world.
   - The datasets are too big for most laptops.
2. Game of Thrones: https://github.com/jeffreylancaster/game-of-thrones

Game of Thrones – Conceptual Model

Add shapes in Lucidchart ...



- With a little
  - Data exploration
  - Common sense
  - Judgment/experience
- I came to this conceptual model.
  - Attributes unspecified
  - Relationship required/cardinality unspecified.

  A little more exploration is needed.

Sample Projects

- Walkthrough of Web Apps:
  - /Users/donaldferguson/Dropbox/000/000-A-Current-Examples/W4111-FastAPI-IMDB-Solution
  - /Users/donaldferguson/Dropbox/000/000-A-Current-Examples/current-dashboard

- Data Engineering:
  - /Users/donaldferguson/Dropbox/000/000-Columbia/2024-Spring/W4111-Intro-to-Databases-Spring-2024/Lectures/s-2024-Lecture-3/More-Data-Engineering.ipynb
  - And others.

- Sample Notebook on GoT.