Database Design

Fundamentals of a good database design

Black Raven (James Ng)
17 Apr 2021 · 31 min read

4 hours 13 Videos 52 Exercises 19,227 Participants 4,150 XP

This is a memo to share what I have learnt in Database Design, capturing the learning objectives as well as my personal notes. The course is taught by Lis Sulmont from DataCamp, and it includes 4 chapters:

Chapter 1. Processing, Storing, and Organizing Data
Chapter 2. Database Schemas and Normalization
Chapter 3. Database Views
Chapter 4. Selecting

A good database design is crucial for a high-performance application. Just like you wouldn't start building a house without the benefit of a blueprint, you need to think about how your data will be stored beforehand.

Taking the time to design a database saves time and frustration later on, and a well-designed database ensures ease of access and retrieval of information. While choosing a design, a lot of considerations have to be accounted for.

In this course, you'll learn how to process, store, and organize data in an efficient way. You'll see how to structure data through normalization and present your data with views. Finally, you'll learn how to manage your database and all of this will be done on a variety of datasets from book sales, car rentals, to music reviews.

Chapter 1. Processing, Storing, and Organizing Data

Start your journey into database design by learning about the two approaches to data processing, OLTP and OLAP. In this first chapter, you'll also get familiar with the different forms data can be stored in and learn the basics of data modeling.

OLTP and OLAP

How should we organize and manage data?

- **Schemas:** *How should my data be logically organized?*
- **Normalization:** *Should my data have minimal dependency and redundancy?*
- **Views:** *What joins will be done most often?*
- **Access control:** *Should all users of the data have the same level of access*
- **DBMS:** *How do I pick between all the SQL and noSQL options?*
- and more!

OLTP

Online **T**ransaction **P**rocessing

OLAP

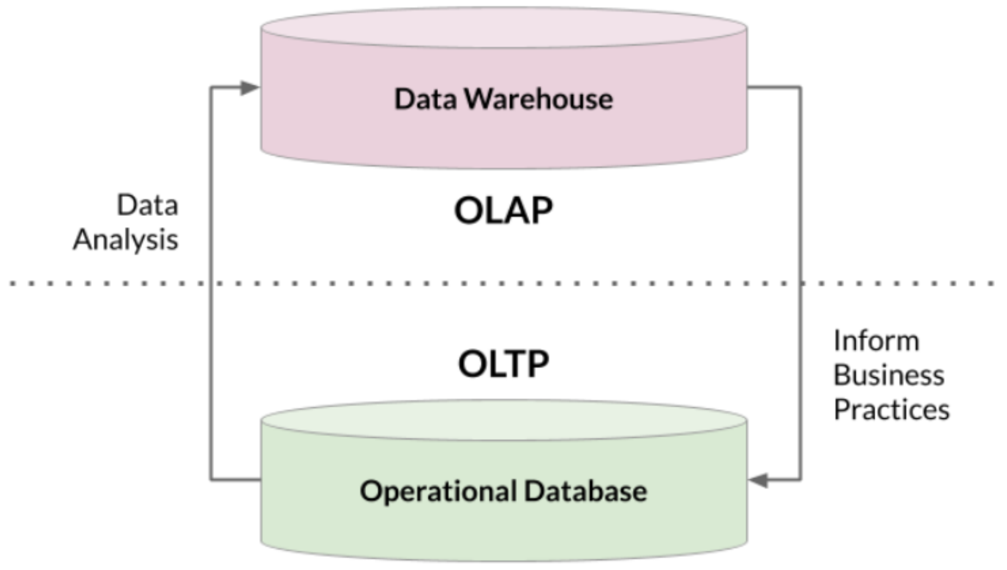
Online **A**nalytical **P**rocessing

- Find the price of a book
- Update latest customer transaction
- Keep track of employee hours

- Calculate books with best profit margin
- Find most loyal customers
- Decide employee of the month

	OLTP	OLAP
Purpose	support daily transactions	report and analyze data
Design	application-oriented	subject-oriented
Data	up-to-date, operational	consolidated, historical
Size	snapshot, gigabytes	archive, terabytes
Queries	simple transactions & frequent updates	complex, aggregate queries & limited updates
Users	thousands	hundreds

OLTP systems are used by more people throughout a company and even a company's customers, while OLAP systems are typically used by only analysts and data scientists at a company.



OLAP and OLTP systems work together; in fact, they need each other. OLTP data is usually stored in an operational database that is pulled and cleaned to create an OLAP data warehouse.

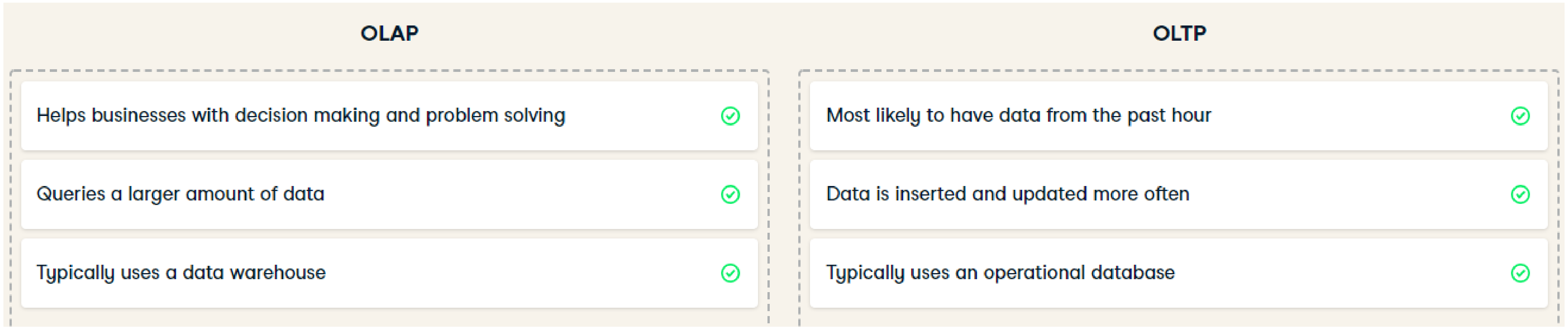
Without transactional data, no analyses can be done in the first place. Analyses from OLAP systems are used to inform business practices and day-to-day activity, thereby influencing the OLTP databases.

Before implementing anything, figure out your business requirements because there are many design decisions you'll have to make. The way you set up your database now will affect how it can be effectively used in the future. Start by figuring out if you need an OLAP or OLTP approach, or perhaps both!

These are the two most common approaches. However, they are not exhaustive, but they are an excellent start to get you on the right path to designing your database.

OLAP vs. OLTP

You should now be familiar with the differences between OLTP and OLAP. In this exercise, you are given a list of cards describing a specific approach which you will categorize between OLAP and OLTP.

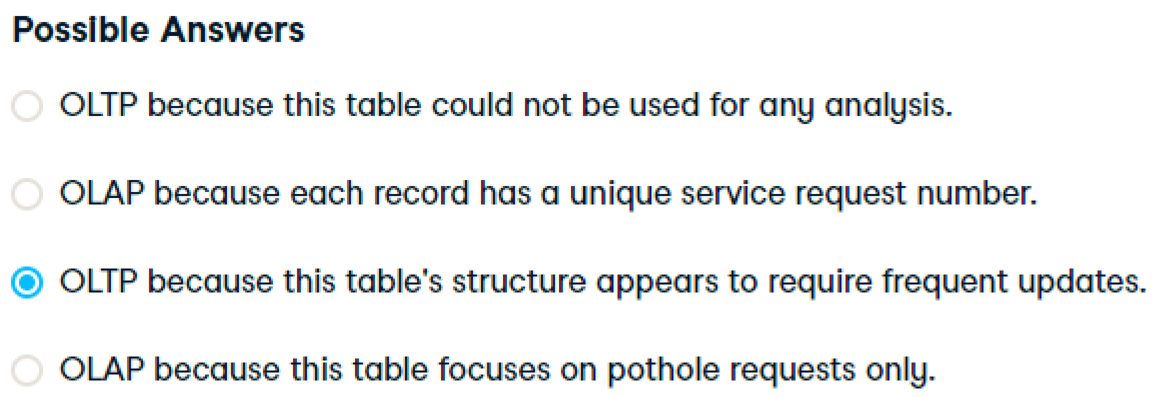


OLAP	OLTP
Helps businesses with decision making and problem solving ✓	Most likely to have data from the past hour ✓
Queries a larger amount of data ✓	Data is inserted and updated more often ✓
Typically uses a data warehouse ✓	Typically uses an operational database ✓

Which is better?

The city of Chicago receives many 311 service requests throughout the day. 311 service requests are non-urgent community requests, ranging from graffiti removal to street light outages. Chicago maintains a data repository of all these services organized by type of requests. In this exercise, `Potholes` has been loaded as an example of a table in this repository. It contains pothole reports made by Chicago residents from the past week.

Explore the dataset. What data processing approach is this larger repository most likely using?

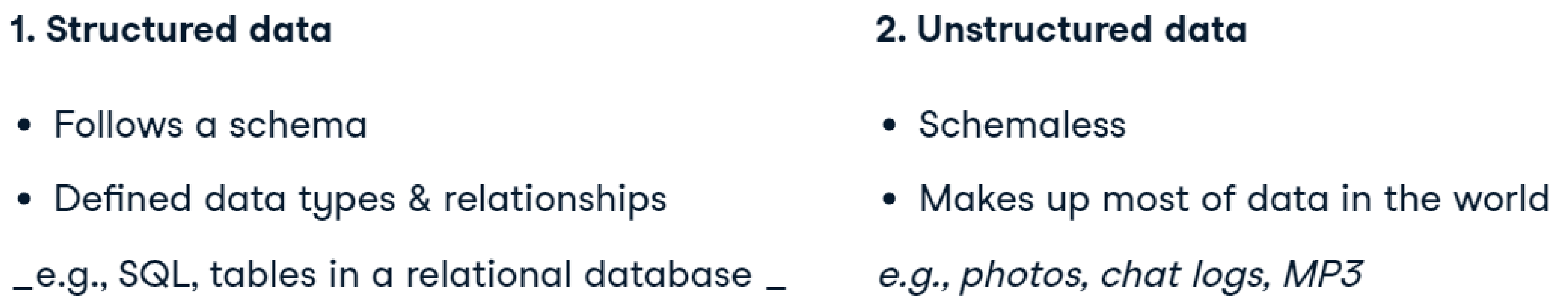


Possible Answers

- OLTP because this table could not be used for any analysis.
- OLAP because each record has a unique service request number.
- [✓ OLTP because this table's structure appears to require frequent updates.]
- OLAP because this table focuses on pothole requests only.

This table probably uses an OLTP approach because it is updated and holds data from the past week.

Storing data



1. Structured data

- Follows a schema
- Defined data types & relationships

_e.g., SQL, tables in a relational database _

2. Unstructured data

- Schemaless
- Makes up most of data in the world

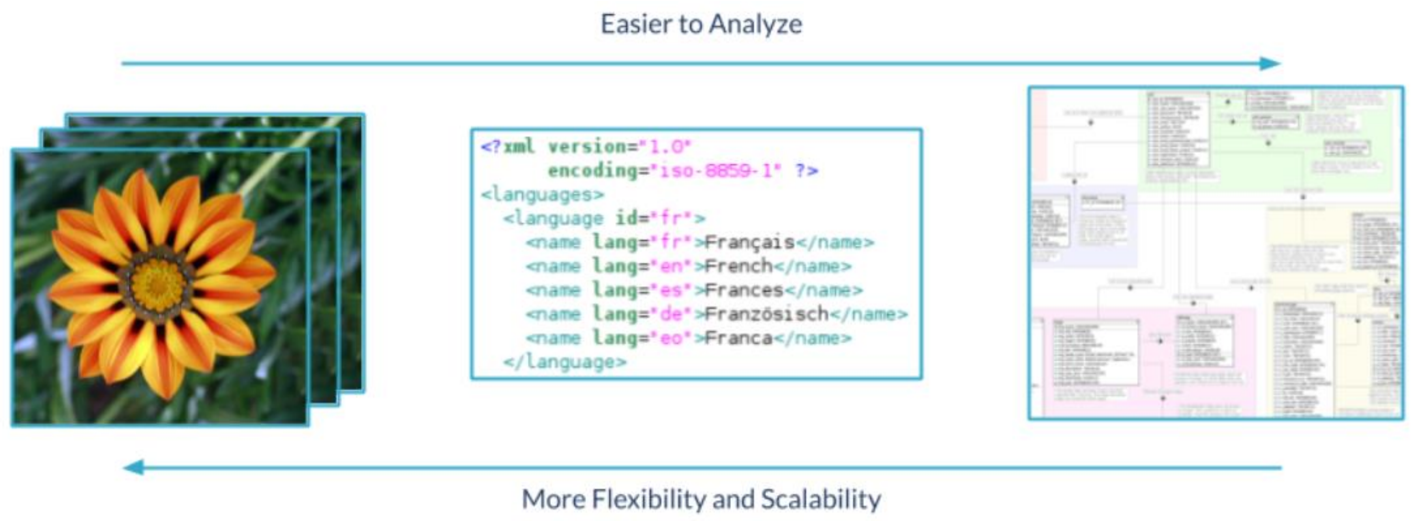
e.g., photos, chat logs, MP3

3. Semi-structured data

- Does not follow larger schema
- Self-describing structure

e.g., NoSQL, XML, JSON

```
# Example of a JSON file
"user": {
     "profile_use_background_image": true,
     "statuses_count": 31,
     "profile_background_color": "C0DEED",
     "followers_count": 3066,
```



```
<?xml version="1.0"
      encoding="iso-8859-1" ?>
<languages>
  <language id="fr">
    <name lang="fr">Français</name>
    <name lang="en">French</name>
    <name lang="es">Frances</name>
    <name lang="de">Französisch</name>
    <name lang="eo">Franca</name>
  </language>
```

- **Traditional databases**
  - For storing real-time relational structured data ? **OLTP**
- **Data warehouses**
  - For analyzing archived structured data ? **OLAP**
- **Data lakes**
  - For storing data of all structures = flexibility and scalability
  - For analyzing **big data**

Traditional **databases** generally follow relational schemas. Operational databases, which are used for OLTP, are an example of traditional databases. Decades ago, traditional databases used to be enough for data storage. Then as data analytics took off, data **warehouses** were popularized for OLAP approaches. And, now in the age of big data, we need to analyze and store even more data, which is where the **data lake** comes in. The term "traditional databases" is used because many people consider data warehouses and lakes to be a type of database.

Data warehouses are optimized for read-only analytics. They combine data from multiple sources and use massively parallel processing for faster queries. In their database design, they typically use dimensional modeling and a denormalized schema. Amazon, Google, and Microsoft all offer data warehouse solutions, known as Redshift, Big Query, and Azure SQL Data Warehouse, respectively. A **data mart** is a subset of a data warehouse dedicated to a specific topic. Data marts allow departments to have easier access to the data that matters to them.

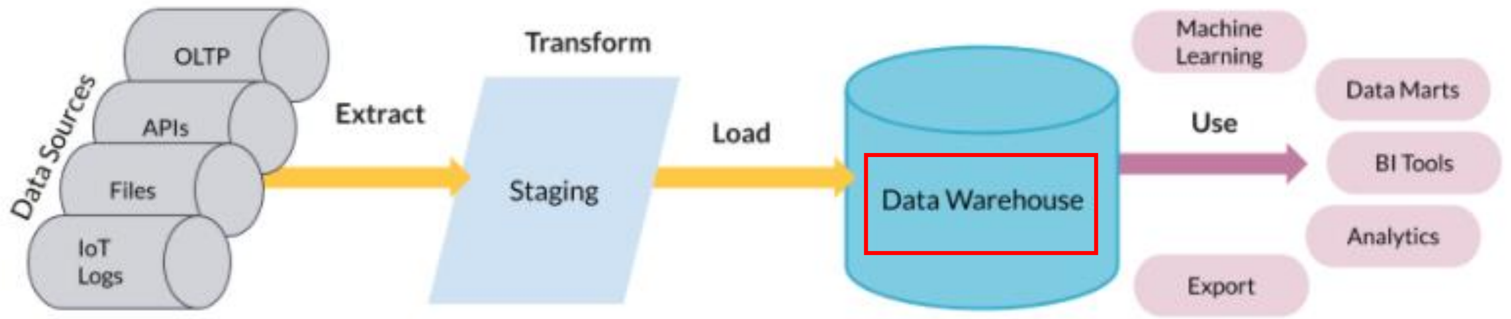
Warehouses and traditional databases are classified as schema-on-write because the schema is predefined.

Traditional databases and warehouses can store unstructured data, but not cost-effectively. Data Lake storage is cheaper because it uses object storage as opposed to the traditional block or file storage. This allows massive amounts of data to be stored effectively of all types, from streaming data to operational databases. Lakes are massive because they store all the data that might be used. Data lakes are often petabytes in size - that's 1,000 terabytes! Unstructured data is the most scalable, which permits this size. Lakes are schema-on-read, meaning the schema is created as data is read.

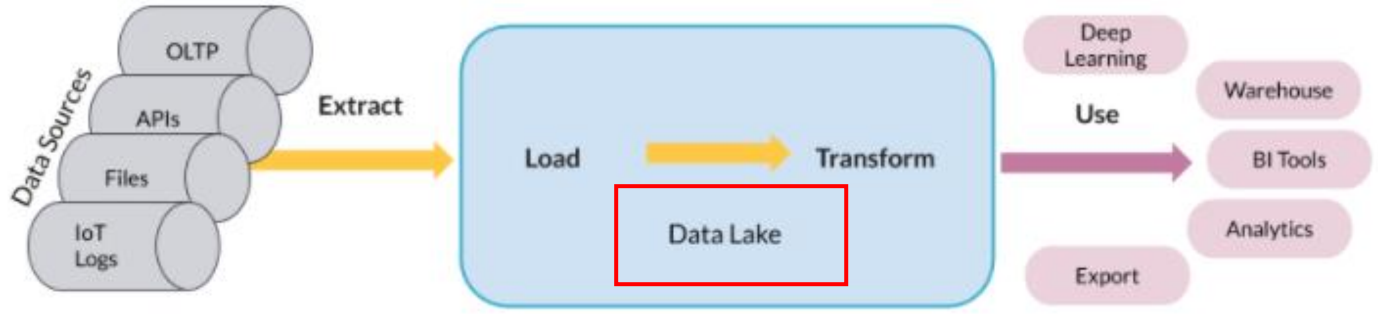
Data lakes have to be organized and cataloged well; otherwise, it becomes an aptly named "data swamp." Data lakes aren't only limited to storage. It's becoming popular to run analytics on data lakes. This is especially true for tasks like deep learning and data discovery, which needs a lot of data that doesn't need to be that "clean."

Difference between ETL and ELT
When we think about where to store data, we have to think about how data will get there and in what form. and Extract Load Transform are two different approaches for describing data flows. They get into the intricacies of building data pipelines, which we will not get into.

ETL



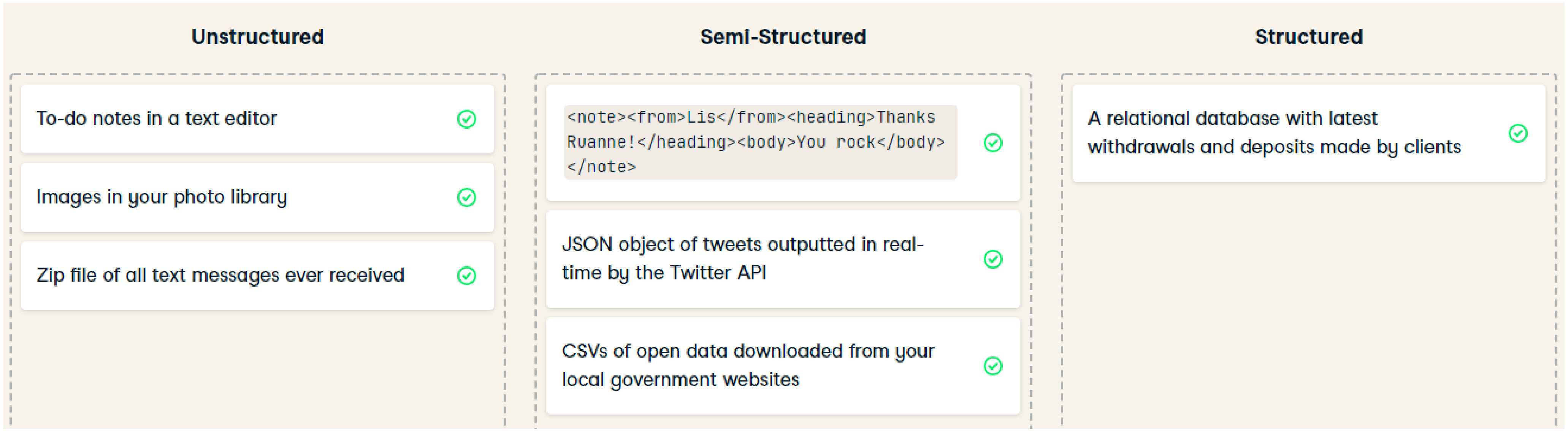
ELT



Extract, Transform, Load or Extract, Load, Transform
ETL is the more traditional approach for warehousing and smaller-scale analytics. But, **ELT** has become common with big data projects. In ETL, data is transformed before loading into storage - usually to follow the storage's schema, as is the case with warehouses.
In ELT, the data is stored in its native form in a storage solution like a data lake. Portions of data are transformed for different purposes, from building a data warehouse to doing deep learning.

Name that data type!

In the previous video, you learned about structured, semi-structured, and unstructured data. Structured data is the easiest to analyze because it is organized and cleaned. On the other hand, unstructured data is schemaless, but scales well. In the middle we have semi-structured data for everything in between.



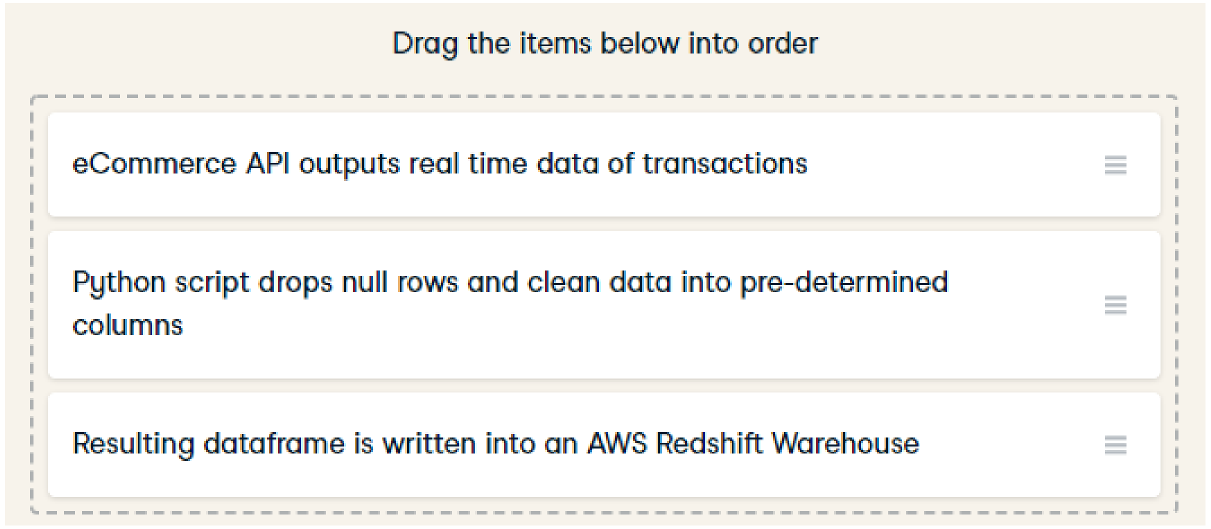
Unstructured	Semi-Structured	Structured
To-do notes in a text editor ✓	`<note><from>Lis</from><heading>Thanks Ruanne!</heading><body>You rock</body></note>` ✓	A relational database with latest withdrawals and deposits made by clients ✓
Images in your photo library ✓	JSON object of tweets outputted in real-time by the Twitter API ✓	
Zip file of all text messages ever received ✓	CSVs of open data downloaded from your local government websites ✓	

From these real-life examples, can you see why unstructured data is easier to scale than structured data?

Ordering ETL Tasks

You have been hired to manage data at a small online clothing store. Their system is quite outdated because their only data repository is a traditional database to record transactions.

You decide to upgrade their system to a data warehouse after hearing that different departments would like to run their own business analytics. You reason that an ELT approach is unnecessary because there is relatively little data (< 50 GB).

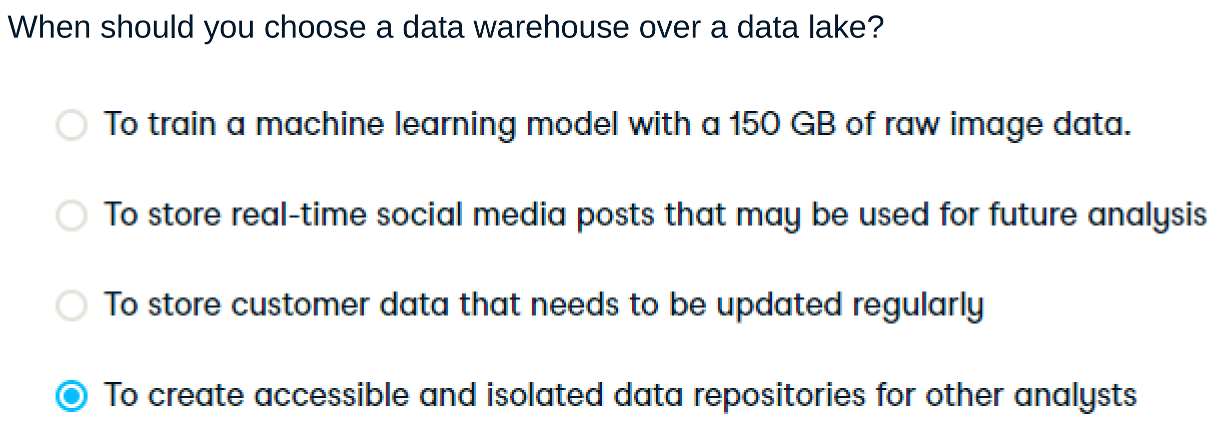


Drag the items below into order

- eCommerce API outputs real time data of transactions
- Python script drops null rows and clean data into pre-determined columns
- Resulting dataframe is written into an AWS Redshift Warehouse

In ETL, raw data is cleaned before being stored. This makes it accessible and ready to use.

Recommend a storage solution



When should you choose a data warehouse over a data lake?

- To train a machine learning model with a 150 GB of raw image data.
- To store real-time social media posts that may be used for future analysis
- To store customer data that needs to be updated regularly
- [✓ To create accessible and isolated data repositories for other analysts]

Analysts will appreciate working in a data warehouse more because of its organization of structured data that make analysis easier.

Database design

Database design determines how data is logically stored. There are two important concepts to know when it comes to database design:

- **Database Models** – high-level specifications for database structure. The relational model, which is the most popular, is the model used to make relational databases. It defines rows as records and columns as attributes. It calls for rules such as each row having unique keys.
- **Database Schemas** – a database's blueprint, ie, the implementation of the database model. It takes the logical structure more granularly by defining the specific tables, fields, relationships, indexes, and views a database will have. Schemas must be respected when inserting structured data into a relational database.

Process of creating a *data model* for the data to be stored

1. Conceptual data model: describes entities, relationships, and attributes

- *Tools:* data structure diagrams, e.g., entity-relational diagrams and UML diagrams

2. Logical data model: defines tables, columns, relationships

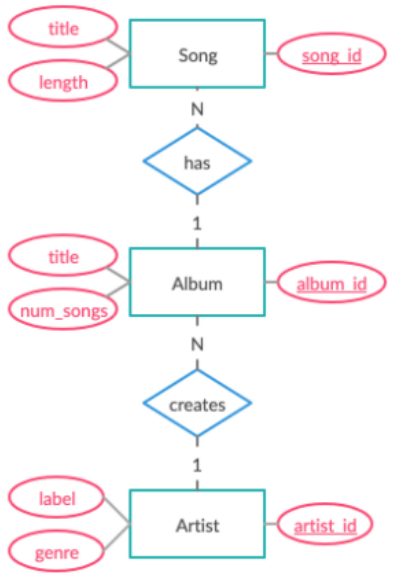
- *Tools:* database models and schemas, e.g., relational model and star schema

3. Physical data model: describes physical storage

- *Tools*: partitions, CPUs, indexes, backup systems and tablespaces

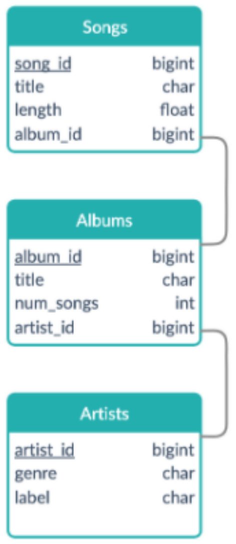
These three levels of a data model ensure consistency and provide a plan for implementation and use.

Conceptual - ER diagram



Entities, relationships, and attributes

Logical - schema



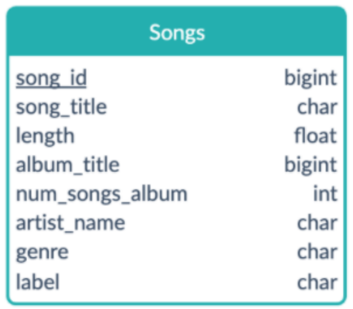
Songs	
song_id	bigint
title	char
length	float
album_id	bigint

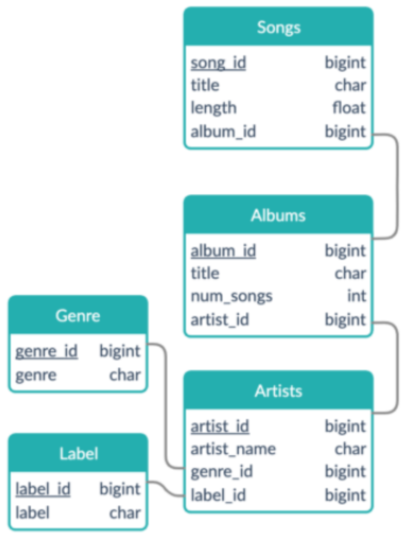
Albums	
album_id	bigint
title	char
num_songs	int
artist_id	bigint

Artists	
artist_id	bigint
genre	char
label	char

Fastest conversion: entities become the tables

Other database design options

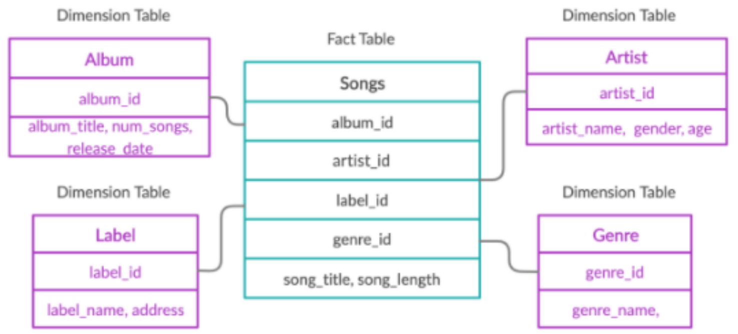




Determining tables

Dimensional modeling is an adaptation of the relational model specifically for data warehouses. It's optimized for **OLAP** type of queries that aim to analyze rather than update. To do this, it uses the star schema. The schema of a dimensional model tends to be easy to interpret and extend.

Elements of dimensional modeling



Fact tables

- Decided by business use-case
- Holds records of a metric
- Changes regularly
- Connects to dimensions via foreign keys

Dimension tables

- Holds descriptions of attributes
- Does not change as often

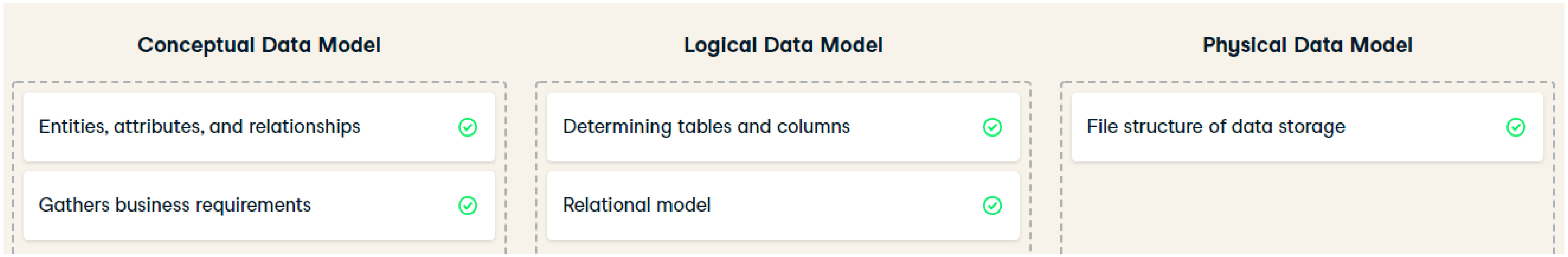
Organize by:

- What is being analyzed?
- How often do entities change?

The records in fact tables often change as new songs get inserted. Albums, labels, artists, and genres will be shared by more than one song - hence records in dimension tables won't change as much.

Classifying data models

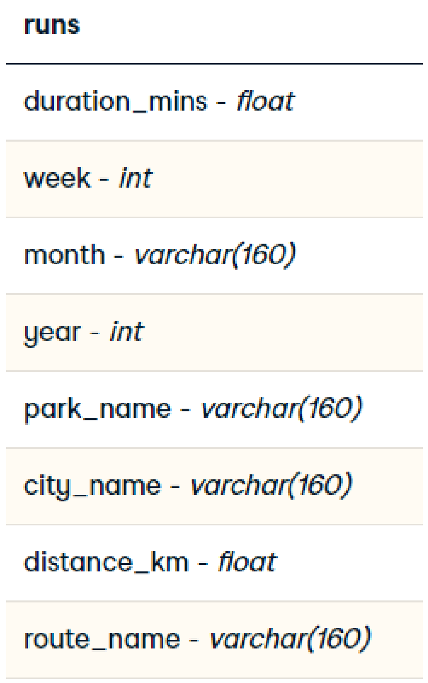
In the previous video, we learned about three different levels of data models: conceptual, logical, and physical.



Conceptual Data Model	Logical Data Model	Physical Data Model
Entities, attributes, and relationships ✓	Determining tables and columns ✓	File structure of data storage ✓
Gathers business requirements ✓	Relational model ✓	

Deciding fact and dimension tables

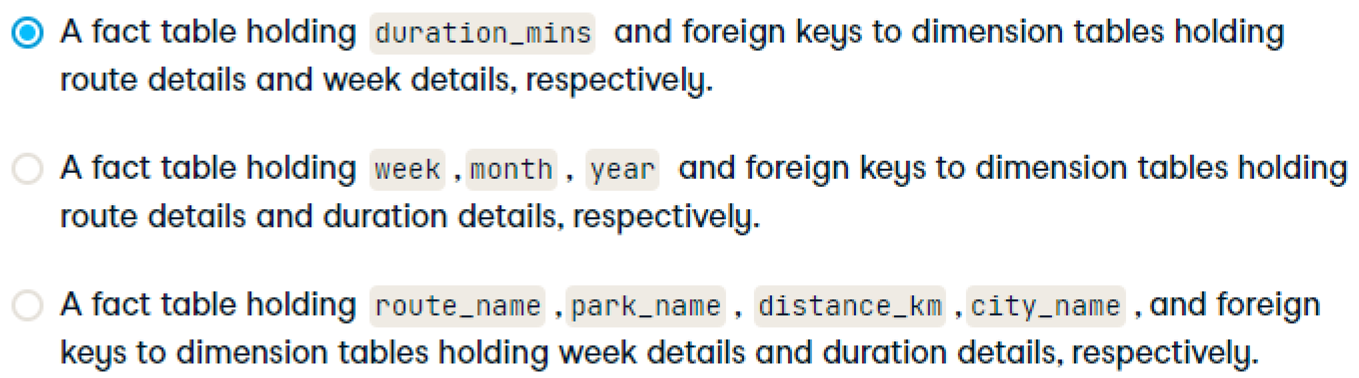
Imagine that you love running and data. It's only natural that you begin collecting data on your weekly running routine. You're most concerned with tracking how long you are running each week. You also record the route and the distances of your runs. You gather this data and put it into one table called `Runs` with the following schema:



runs
duration_mins - *float*
week - *int*
month - *varchar(160)*
year - *int*
park_name - *varchar(160)*
city_name - *varchar(160)*
distance_km - *float*
route_name - *varchar(160)*

After learning about dimensional modeling, you decide to restructure the schema for the database. `Runs` has been pre-loaded for you.

Out of these possible answers, what would be the best way to organize the fact table and dimensional tables?



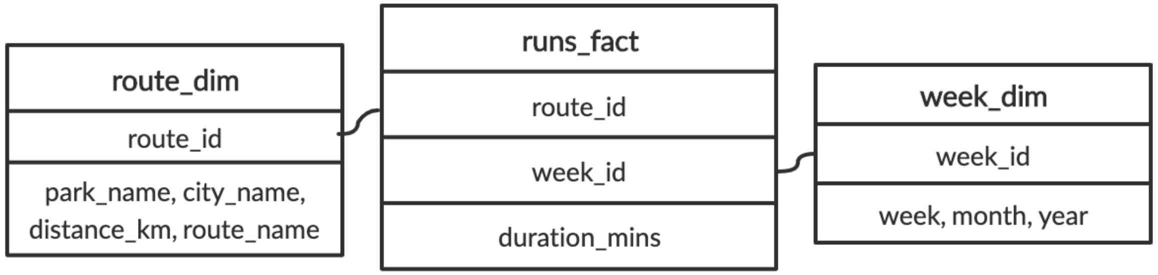
- [✓ A fact table holding `duration_mins` and foreign keys to dimension tables holding route details and week details, respectively.]
- A fact table holding `week`, `month`, `year` and foreign keys to dimension tables holding route details and duration details, respectively.
- A fact table holding `route_name`, `park_name`, `distance_km`, `city_name`, and foreign keys to dimension tables holding week details and duration details, respectively.

```sql
-- Create a route dimension table
CREATE TABLE route(
    route_id INTEGER PRIMARY KEY,
    route_name VARCHAR(160) NOT NULL,
    park_name VARCHAR(160) NOT NULL,
    distance_km FLOAT NOT NULL,
    city_name VARCHAR(160) NOT NULL
);
-- Create a week dimension table
CREATE TABLE ___(
    week_id INTEGER PRIMARY KEY,
    week integer NOT NULL,
    month VARCHAR(160) NOT NULL,
    year integer NOT NULL
);
```

The primary keys route_id and week_id you created will be foreign keys in the fact table.

Querying the dimensional model

Here it is! The schema reorganized using the dimensional model:



Let's try to run a query based on this schema. How about we try to find the number of minutes we ran in July, 2019? We'll break this up in two steps. First, we'll get the total number of minutes recorded in the database. Second, we'll narrow down that query to `week_id`'s from July, 2019.

Chapter 2. Database Schemas and Normalization

In this chapter, you will take your data modeling skills to the next level. You'll learn to implement star and snowflake schemas, recognize the importance of normalization and see how to normalize databases to different extents.

Chapter 3. Database Views

Get ready to work with views! In this chapter, you will learn how to create and query views. On top of that, you'll master more advanced capabilities to manage them and end by identifying the difference between materialized and non-materialized views.

Chapter 4. Database Management

This final chapter ends with some database management-related topics. You will learn how to grant database access based on user roles, how to partition tables into smaller pieces, what to keep in mind when integrating data, and which DBMS fits your business needs best.

Course completed!

Recap topics covered:

- Basics

Next Steps

- Basics

Happy learning!