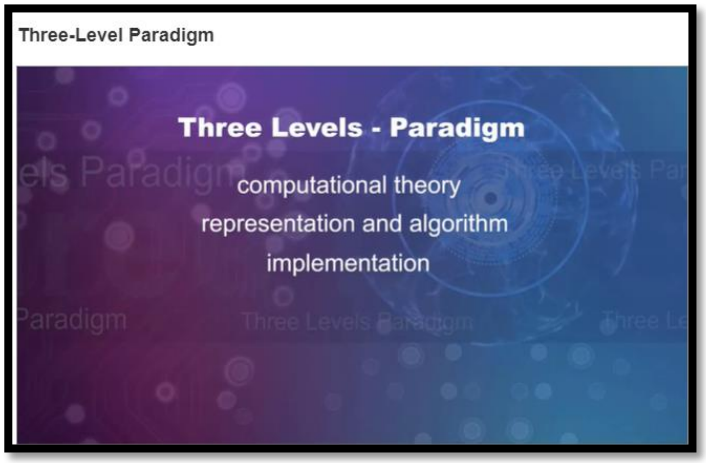
DAILY ASSESSMENT

Date:	**15/07/2020**	**Name:**	**Davis S. Patel**
Course:	**Computer Vision Basics**	**USN:**	**4AL16EC045**
Topic:	**Week 3**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

FORENOON SESSION DETAILS

Image of session

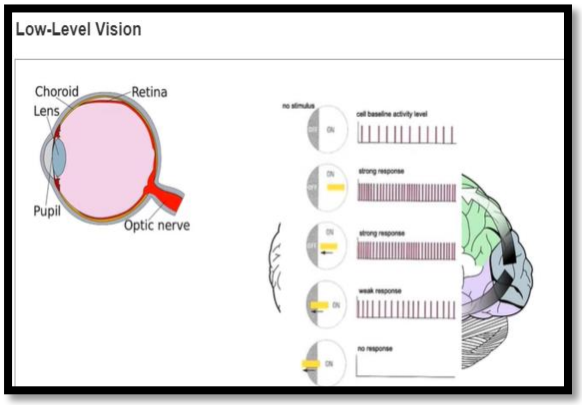

Three-Level Paradigm

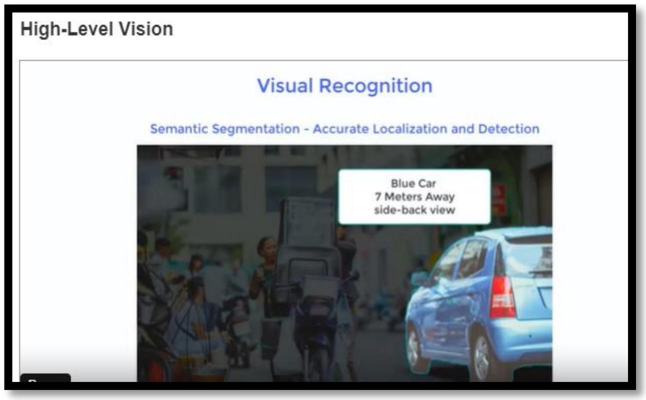
Three Levels - Paradigm

computational theory

representation and algorithm

implementation


Low-Level Vision


High-Level Vision

Visual Recognition

Semantic Segmentation - Accurate Localization and Detection

REPORT –

In recent work in the theoretical foundations of cognitive science, it has become commonplace to separate three distinct levels of analysis of information-processing systems. David Marr (1982) has dubbed the three levels the computational, the algorithmic, and the implementation; Zenon Pylyshyn (1984) calls them the semantic, the syntactic, and the physical; and textbooks in cognitive psychology sometimes call them the levels of content, form, and medium (e.g. Glass, Holyoak, and Santa 1979).

Levels of organization

It has become a central tenet of the current conventional wisdom in the philosophy of science that complex systems are to be seen as typically having multiple levels of organization. The standard model of the multiple levels of a complex system is a rough hierarchy, with the components at each ascending level being some kind of composite made up of the entities present at the next level down. We thus often have explanations of a system's behavior at higher (coarser-grained) and lower (finer-grained) levels. The behavior of a complex system -- a particular organism, say -- might then be explained at various levels of organization, including (but not restricted to) ones which are biochemical, cellular, and psychological. And similarly, a given computer can be analyzed and its behavior explained by characterizing it in terms of the structure of its component logic gates, the machine language program it's running, the Lisp or Pascal program it's running, the accounting task it's performing, and so on.

Higher-level explanations allow us to explain as a natural class things with different underlying physical structures -- that is, types which are *multiply realizable*. [See, e.g., Fodor (1974), Pylyshn (1984), esp. chapter 1, and Kitcher (1984), esp. pp.343-6, for discussions of this central concept.] Thus, we can explain generically how transistors, resistors, capacitors, and power sources interact to form a kind of amplifier independent of considerations about the various kinds of materials composing these parts, or account for the relatively independent assortment of genes at meiosis without concerning ourselves with the exact underlying chemical mechanisms. Similar points can be made for indefinitely many cases: how an adding machine works, an internal combustion engine, a four-chambered heart, and so on.

Function and context

The idea of specifying the components of a larger system in terms of their overall functional role with respect to that embedding system is a central aspect of our general framework of explanation which plays several key roles in the understanding of complex systems. The functional properties of parts of complex systems are *context-dependent* properties -- ones which depend on occurring in the right context, and not just on the local and intrinsic properties of the particular event or object itself. The phenomenon of context-dependence of course shows up often in the taxonomies of various sciences. Examples abound: The position of a given DNA sequence with respect to the rest of the genetic material is critical to its status as a *gene*; type-identical DNA sequences at different loci can play different hereditary roles -- be different genes, if you like. So for a particular DNA sequence to be, say, a brown-eye gene, it must be in an appropriate position on a particular chromosome. Similarly for a given action of a computer's CPU, such as storing the contents of internal register A at the memory location whose address is contained in register X: Two instances of that very same action might, given different positions in a program, differ completely in terms of their functional properties at the higher level: At one place in a program, it might be "set the carry digit from the last addition", and at another, "add the new letter onto the current line of text". And for mechanical systems -- e.g. a carburetor: The functional properties of being a choke or being a throttle are context-dependent. The very same physically characterized air flow valve can be a choke in one context (i.e. when it occurs above the fuel jets) and a throttle in another (when it occurs below the jets); whether a given valve is a choke or a throttle depends on its surrounding context. By "contextualizing" objects in this way we shift from a categorization of them in terms of local and intrinsic properties to their context-dependent functional ones.

The three levels

In chapter 1.2 of *Vision*, David Marr presents his variant on the "three levels" story. His summary of "the three levels at which any machine carrying out an information-processing task must be understood":

- *Computational theory*: What is the goal of the computation, why is it appropriate, and what is the logic of the strategy by which it can be carried out?
- *Representation and algorithm*: How can this computational theory be implemented? In particular, what is the representation for the input and output, and what is the algorithm for the transformation?
- *Hardware implementation*: How can the representation and algorithm be realized physically?

Top-down vs. bottom-up approaches

Generally by the mid-1980s the top-down paradigm of symbolic AI was being questioned while distributed and bottom-up models of mind were gaining popularity. In computation two major fields developed, connectionism and evolutionary computing. Other bottom-up trends in AI have been, situated cognition (with varied threads including anthropology and robotics) and distributed AI. Shades of the rationalist-empiricist debate are seen here.

The advantages and disadvantages of the t-d and b-u approaches in AI are complementary.

Top-down (aka symbolic) approach

Hierarchically organized (top-down) architecture

All the necessary knowledge is pre-programmed, i.e. already present - in the knowledge base.

Analysis/ computation involves creating, manipulating and linking symbols (hence propositional and predicate- calculus approach).

``Serial executive'' might be seen as the conscious rule-interpreter which acts on the parallel-processing unconscious intuitive processor.

Thus the program performs better at relatively high-level tasks such as language processing aka NLP - it is consistent with currently accepted theories of language acquisition which assume some high-level modularity. But how well are subtleties of language handled.

Bottom-up approach - e.g. Neural networks

Models are built from simple components connected in a network.

Relatively simple abstract program consisting of learning cycles.

Program builds its own (distributed) ``knowledge base'' and ``commonsense assertions''.

Normally done with parallel processing, or more commonly with data structures simulating parallel processing, such as neural networks.

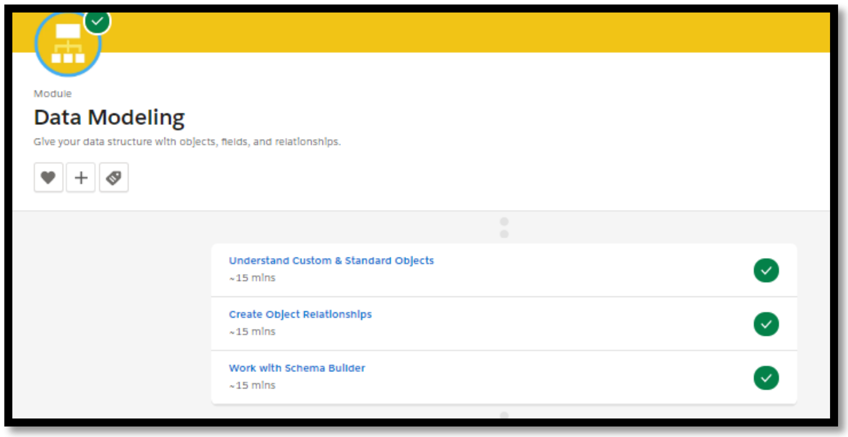
‘High-level' vision lacks a single, agreed upon definition, but it might usefully be defined as those stages of visual processing that transition from analyzing local image structure to analyzing structure of the external world that produced those images. Much work in the last several decades has focused on object recognition as a framing problem for the study of high-level visual cortex, and much progress has been made in this direction. This approach presumes that the operational goal of the visual system is to read-out the identity of an object (or objects) in a scene, in spite of variation in the position, size, lighting and the presence of other nearby objects. However, while object recognition as a operational framing of high-level is intuitive appealing, it is by no means the only task that visual cortex might do, and the study of object recognition is beset by challenges in building stimulus sets that adequately sample the infinite space of possible stimuli.

DAILY ASSESSMENT

Date:	**15/07/2020**	**Name:**	**Davis S. Patel**
Course:	**Salesforce Developer**	**USN:**	**4AL16EC045**
Topic:	**Data Modelling**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

AFTERNOON SESSION DETAILS

Image of Session



Module

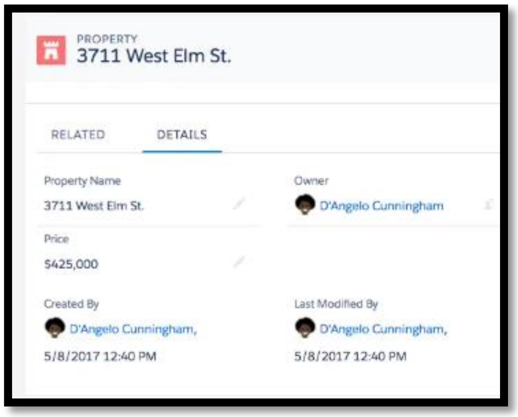
Data Modeling

Give your data structure with objects, fields, and relationships.

Understand Custom & Standard Objects
~15 mins

Create Object Relationships
~15 mins

Work with Schema Builder
~15 mins



PROPERTY
3711 West Elm St.

RELATED DETAILS

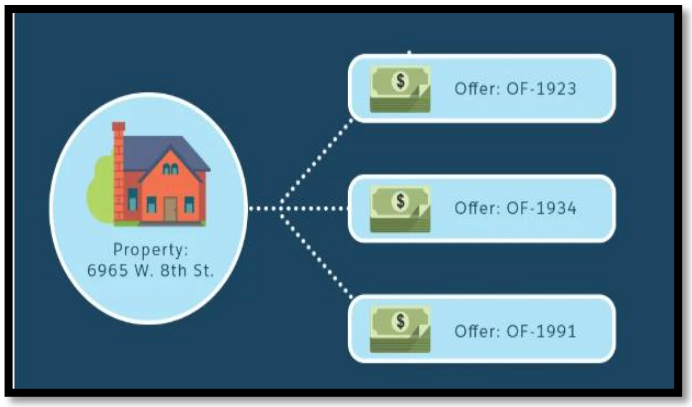
Property Name: 3711 West Elm St.

Owner: D'Angelo Cunningham

Price: $425,000

Created By: D'Angelo Cunningham, 5/8/2017 12:40 PM

Last Modified By: D'Angelo Cunningham, 5/8/2017 12:40 PM



REPORT -

DreamHouse is a realty company that provides a way for customers to shop for homes and contact real estate agents online. DreamHouse brokers use some of Salesforce’s standard functionality, like contacts and leads, to track home buyers.But when it comes to selling houses, there are a lot more things they want to track. For example, Salesforce doesn’t include a standard way to track properties.

A data model is more or less what it sounds like. It’s a way to model what database tables look like in a way that makes sense to human.

Salesforce supports several different types of objects. There are standard objects, custom objects, external objects, platform events, and BigObjects. In this module, we focus on the two most common types of objects: standard and custom.

Standard objects are objects that are included with Salesforce. Common business objects like Account, Contact, Lead, and Opportunity are all standard objects.

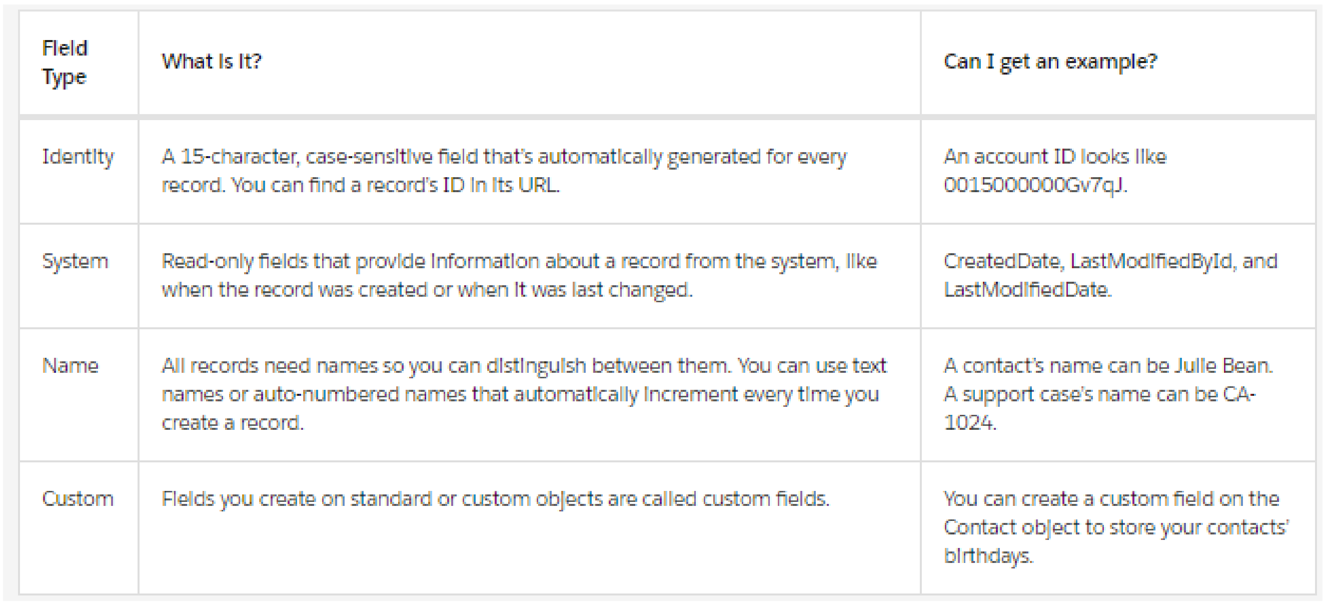
Custom objects are objects that you create to store information that’s specific to your company or industry. For DreamHouse, D’Angelo wants to build a custom Property object that stores information about the homes his company is selling.

Objects are containers for your information, but they also give you special functionality. For example, when you create a custom object, the platform automatically builds things like the page layout for the user interface.

Identity, system, and name fields are standard on every object in Salesforce. Each standard object also comes with a set of prebuilt, standard fields. You can customize standard objects by adding custom fields, and you can add custom fields to your custom objects.

Every field has a data type. A data type indicates what kind of information the field stores. Salesforce supports a bunch of different data types, but here are a few you’ll run into.

- **Checkbox**—for fields that are a simple “yes” or “no,” a checkbox field is what you want.
- **Date or Datetime**—these field types represent dates or date/time combinations, like birthdays or sales milestones.
- **Formula**—this special field type holds a value that’s automatically calculated based on a formula that you write



Field Type	What Is It?	Can I get an example?
Identity	A 15-character, case-sensitive field that’s automatically generated for every record. You can find a record’s ID in its URL.	An account ID looks like 0015000000Gv7qJ.
System	Read-only fields that provide information about a record from the system, like when the record was created or when it was last changed.	CreatedDate, LastModifiedById, and LastModifiedDate.
Name	All records need names so you can distinguish between them. You can use text names or auto-numbered names that automatically increment every time you create a record.	A contact’s name can be Julie Bean. A support case’s name can be CA-1024.
Custom	Fields you create on standard or custom objects are called custom fields.	You can create a custom field on the Contact object to store your contacts’ birthdays.

The Wide World of Object Relationships

There are two main types of object relationships: lookup and master-detail.

Lookup Relationships

In our Account to Contact example above, the relationship between the two objects is a **lookup relationship**. A lookup relationship essentially links two objects together so that you can “look up” one object from the related items on another object.

Lookup relationships can be one-to-one or one-to-many. The Account to Contact relationship is one-to-many because a single account can have many related contacts. For our DreamHouse scenario, you could create a one-to-one relationship between the Property object and a Home Seller object.

Master-Detail Relationships

While lookup relationships are fairly casual, **master-detail relationships** are a bit tighter. In this type of relationship, one object is the master and another is the detail. The master object controls certain behaviors of the detail object, like who can view the detail's data.

For example, let's say the owner of a property wanted to take their home off the market. DreamHouse wouldn't want to keep any offers made on that property. With a master-detail relationship between Property and Offer, you can delete the property and all its associated offers from your system.

Create an Object with Schema Builder

Schema Builder is great for visualization, but you can also use it to customize your data model. For example, you can manage the permissions for your custom fields directly in Schema Builder. Just right-click the field name and click **Manage Field Permissions.**

You can also create objects using Schema Builder. If you prefer, you can create objects in this visual interface if you're designing your system and want to be able to revise all your customizations on the spot.

Create Fields with Schema Builder

Creating fields with Schema Builder is just like creating objects.

1. From the Elements tab, choose a field type and drag it onto the object you just created. Notice that you can create relationship fields, formula fields, and normal fields in Schema Builder.

2. Fill out the details about your new field.

3. Click **Save**.