David Chappell

INTRODUCING AZURE SEARCH

Sponsored by Microsoft Corporation

Contents

Understanding Azure Search

How should an application let its users interact with data? The answer is obvious: It depends. Sometimes all that's needed is to expose a familiar web or mobile user interface, then let the user click her way to what she needs.

More and more often, though, users expect to interact with data through search. People love search; it's simple and powerful and requires no training to use. And since it's the default style of interaction with the web, search is increasingly expected in every situation. No matter what kind of user interface an application offers, including a search option is probably a good thing.

But providing this option can be challenging for application developers. It's not reasonable to expect every development team to build its own search engine. Even installing and running a commercial search engine can be a lot of work. What's needed is a managed search service that can be used by many different applications, whether they're running in the cloud or on premises. This is exactly what Azure Search offers.

What Azure Search Provides

Azure Search is a managed service running in the public cloud. A development team can create a new instance of the service, then start using it right away. The team doesn't need to install or manage its own search technology.

An application using Azure Search can run on Microsoft Azure, on-premises, or even on some other cloud platform. Wherever it runs, the application will typically rely on a relational or NoSQL database for operational data, such as records of sales, user-generated content, or whatever else the application works with. On Azure, for example, this database might be a managed service such as SQL Database or DocumentDB. It might also be a relational database management system (DBMS) such as SQL Server or a NoSQL store running in an Azure virtual machine. Similarly, an on-premises application might use a relational DBMS or a NoSQL store for its operational data. Whatever the application uses, Azure Search sits alongside this database, providing indexes that can be used to search the operational data. Figure 1 shows a typical scenario.

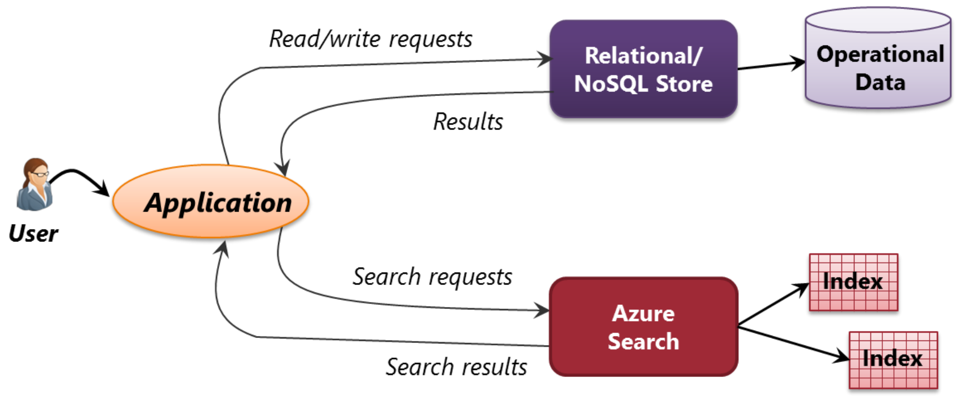


Figure 1: Azure Search helps developers create applications that can search their operational data.

It's important to understand that Azure Search is entirely focused on developers. It can't be used directly by end users. Instead, Azure Search exposes a RESTful interface to applications, so all access to the service is via standard HTTP verbs such as GET and PUT. (To make this interface easier to use, Microsoft provides a .NET SDK that hides the raw RESTful calls.) Also, Azure Search says nothing at all about how an application's user interface (UI) should look. The creators of an application are free to provide any search UI they like.

But what kind of applications need to have their own search service? Internet services such as Bing and Google can be scoped to a single site, so why can't applications on the web just use these? Why should a development team bother to create its own custom search service?

In a word, the answer is *control*. To see why this matters, it's useful to look at some of the key scenarios for which Azure Search was designed. They include the following:

- *E-commerce applications*, such as the web site for an online retailer. Providing a search option for an e-commerce site is essential—users expect it. But the organization that provides this site almost certainly wants to control what information is returned and—especially—the order of those results. Think of an online shoe store, for example, that's currently running a promotion with a particular shoe manufacturer. Suppose that manufacturer is paying the online retailer for this promotion, and so the site's search results need to list this brand of shoes first. Or perhaps the shoe store has lots of a particular style in stock right now that it wants to sell off. Placing the style first in search results can help the retailer achieve this business goal. Owning its own search function has other benefits, too, such as letting this firm see what its customers are searching for that it doesn't currently sell. None of this makes internet search engines any less important; an online retailer should still do whatever it can to direct Google and Bing searches to its site. Once people are there, however, the retailer can benefit from controlling how customers search the site.

- *User-generated content sites*, such as a discussion site for movie buffs. As with e-commerce applications, users expect to be able to navigate this kind of site via search. For the creators of the site, controlling that search once again brings some advantages. As with e-commerce applications, for example, there might be business reasons for returning search results in a particular order. Suppose an online cooking site is sponsored by three large food companies. The site's owners might choose to show recipes that use foods sold by these companies higher in search results. (This might seem cynical—do we really need more pay-for-play sites?—yet it is in fact how much of the internet business works.) And because Azure Search lowers the barrier to entry for creating custom search, an organization doesn't need to realize enormous benefits to justify the effort of doing this.

- *Custom business applications*, such as an employee benefits solution. Traditionally, a line-of-business application is accessed by clicking through its UI until the user finds what he needs. If the application is simple, or if the user knows the application very well, this approach works. But many business applications (maybe even most of them) would be significantly more usable if they provided a search option. Once again, people love search. (Don't you?) Add a search box to an application's UI, then watch how rapidly people start using it. In fact, one way to smooth adoption of a new business application in an organization might be to make sure that it has a search option in its UI.

Is search on its way to becoming a standard part of new applications? Very possibly—it has a lot to offer. But in at least the three scenarios just described, search is clearly useful, which is why Azure Search was designed to address situations like these.

What’s Required to Support a Search UI?

The goal of Azure Search is to help developers create applications that let users access data through search. But what do developers need to do this? What services should Azure Search provide? To answer this question, we need to start by looking at the search process from a user’s point of view.

The most common way for people to access a search service today is through a search box. There are other options, such as map-based searches, but typing text into a box with the standard magnifying glass logo has become the norm. Figure 2 shows an example.

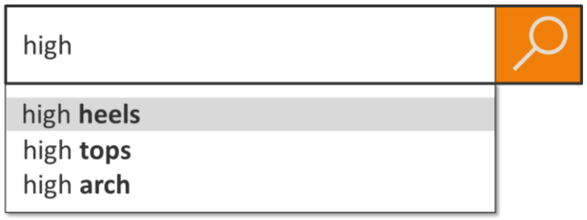
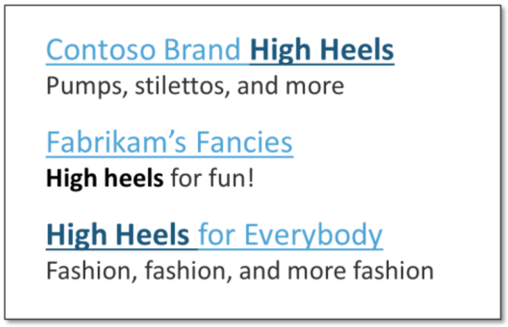


Figure 2: A search box with suggestions is today’s typical UI for issuing searches.

Once again, don’t be confused: Azure Search doesn’t provide this or any other search UI. How a user makes search requests is entirely up to the people building the application. But using this concrete example helps us think about what kind of services Azure Search should provide.

For example, we’ve all come to expect a search box to provide suggestions. The example shown here is for an online shoe store, and so when the user types “high”, she sees suggestions that include “high heels”, “high tops”, and “high arch”. To help the developer provide this, Azure Search includes a service that returns suggestions.

Once the user chooses a phrase to search for, e.g., “high heels”, she gets back search results. Figure 3 shows a simple example of how those results might look.



Contoso Brand **High Heels**
Pumps, stilettos, and more

Fabrikam’s Fancies
High heels for fun!

High Heels for Everybody
Fashion, fashion, and more fashion

Figure 3: Azure Search supports showing results as simple text with bolded search terms.

Here, the search results are returned as simple text. This is sufficient for some applications, and it’s relatively easy for applications to display. Even with such a basic format, though, Azure Search must provide support for specific things.

For example, it's traditional to show the text the user searched for in bold, as this example does. Azure Search makes this easy to do. More important, the search engine needs to let the creators of the application control the order in which results are returned. Azure Search has strong support for this, as described in some detail later.

An online shoe store that showed only text search results, as in Figure 3, would quickly go out of business. Many scenarios, including online retailers and others, require showing search results in a more complex way. Figure 4 shows something closer to how this online retailer might actually return the results of a search for high heels.

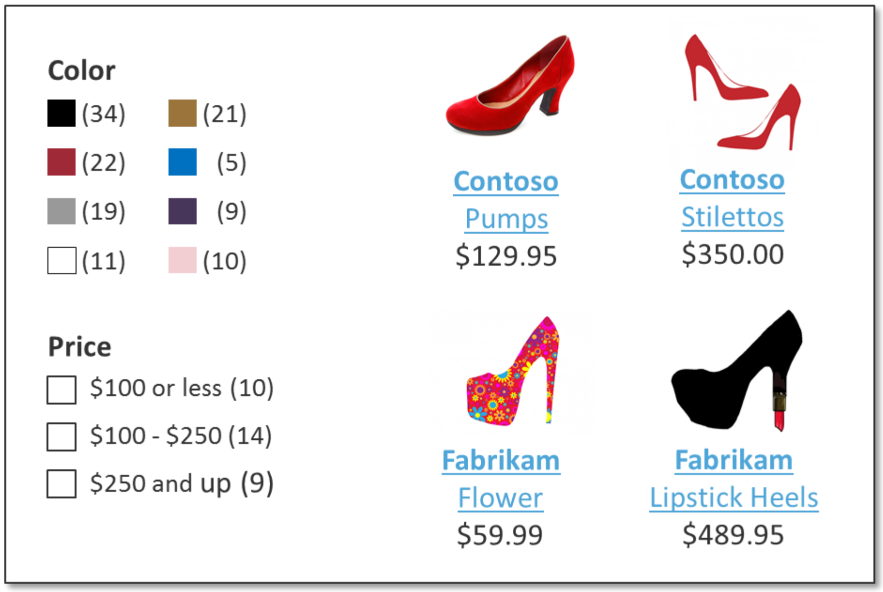


Figure 4: Azure Search also supports showing results in a more complex way.

Creating this kind of UI requires more information than just simple text. For one thing, the user's search must return pictures, giving her an idea of what the shoes look like. And notice the categories shown on the left: Color and Price. These search results show how many items are available in each category, helping with the user's next search. As described later, Azure Search provides a variety of services aimed at helping applications show results like these.

Whether a user's search results are simple or more complex, there's one more thing that's always required: speed. People have no patience for slow searches, largely because we've all been conditioned by fast internet search engines to expect great performance. To help keep an application's users happy, Azure Search provides mechanisms for scaling the service up and down to support the required load.

Meeting user expectations requires all of the things just described (and a few more). The rest of this overview looks more closely at the services Azure Search provides to support this user experience.

Using Azure Search

To start using Azure Search, a developer first creates an instance of the service. Among other things, the developer specifies which Azure datacenter this instance should run in and how much capacity the instance should provide. Once this has been done, an application can begin using the service via the RESTful interface it provides. To get started, an application must first create one or more indexes. Each index contains information that a search request can access, and it's the fundamental data store of Azure Search. Once an index exists, the application can begin issuing searches against it and displaying the results. Finally, the application must periodically update the index as the data it searches on changes. What follows takes a high-level look at each of these steps.

Creating an Index

Getting an index ready to use requires two things:

- Defining the index's schema, including specifying the fields it contains and setting various attributes for each field.
- Populating the index by supplying its initial set of data. Most often, this data will come from the application's operational database, but this isn't required. An index can also hold data from other sources.

Figure 5 illustrates these two steps.

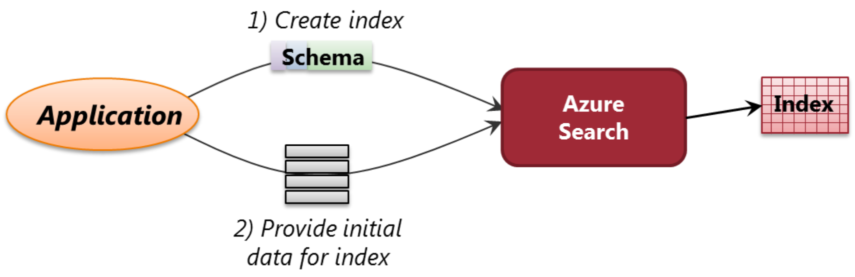


Figure 5: Before an index can be used, it must be created and populated with data.

Notice that the user doesn't appear in this figure. These steps are done entirely by the application, and so the user isn't involved.

Defining an Index's Schema

As part of creating an index, an application supplies a schema expressed using JavaScript Object Notation (JSON). The schema describes the fields this index can contain, each of which has some number of attributes. The attributes a field can have include the following:

- Name: Describes the data this field contains.
- Type: Indicates the type of data in this field. The options include String, Int32, Double, Boolean, and more.
- Searchable: Determines whether a user's search request can access this field.

- Suggestions: Determines whether Azure Search can provide suggestions (such as “high heels” when the user types “high”) for this field. If this is set, an application can call Azure Search regularly while the user is typing into the search box to get suggestions. These suggestions are added to the index by the people who own that index—they’re not created automatically by the search service.

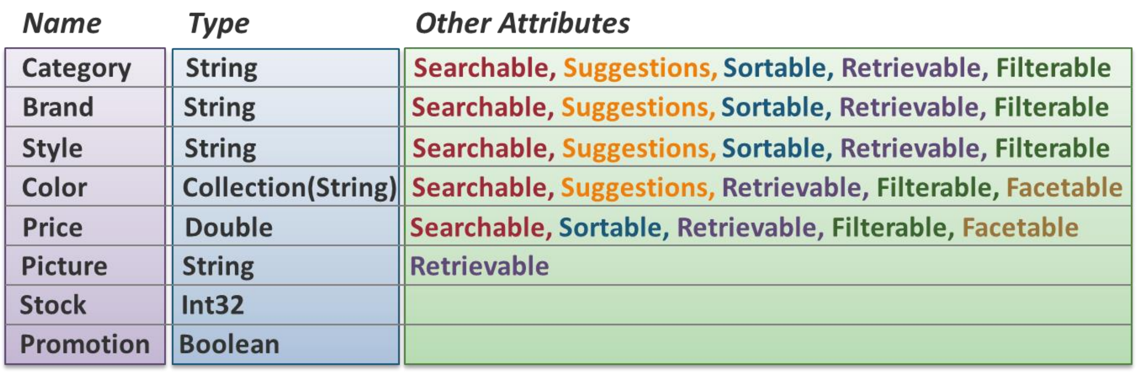
- Sortable: Indicates that search results can be sorted by this field. Some fields, such as a string containing a paragraph of text, might not allow this, since sorting on a paragraph probably wouldn’t make much sense.

- Retrievable: Indicates whether this field can be returned in search results.

- Filterable: Indicates that this field can be used as a filter. For example, if a user wishes to search for “high heels”, the field that contains these search terms must be marked as filterable. This lets Azure Search return only the rows in the index that contain “high heels” in that field.

- Facetable: Indicates whether a search request can return the number of items in the index with a specific characteristic, such as color=red. An application can also request the number of items within a specific range, such as all items whose price is between 100 and 200 euros.

The best way to understand this is to look at a simple example. Suppose you’re creating the public web site for an online shoe retailer, and you’d like to create an Azure Search index that supports the graphical search results shown earlier in Figure 4. Figure 6 shows how that index’s schema might look.



Name	*Type*	*Other Attributes*
Category	String	Searchable, Suggestions, Sortable, Retrievable, Filterable
Brand	String	Searchable, Suggestions, Sortable, Retrievable, Filterable
Style	String	Searchable, Suggestions, Sortable, Retrievable, Filterable
Color	Collection(String)	Searchable, Suggestions, Retrievable, Filterable, Facetable
Price	Double	Searchable, Sortable, Retrievable, Filterable, Facetable
Picture	String	Retrievable
Stock	Int32	
Promotion	Boolean	

Figure 6: Each field in an index has a name, a type, and possibly other attributes.

The first three fields in the index—Category, Brand, and Style—are very similar. They’re all strings, and they have the same set of attributes. The next field, Color, is a bit different. It’s a collection of strings—the same shoes can come in many colors—so it’s not Sortable. (How would you sort a collection?) Also, this field and the next one, Price, are marked as facetable, which means that an application can ask the index for a count of each color or for the number of shoes at various prices. And note that Price doesn’t support suggestions; how much value would there be in making suggestions as you enter a price?

The next field, Picture, is intended to contain a URL that references, say, a JPEG file with an image of this shoe. That URL might point to Azure Blobs or somewhere else. Notice that while this field is retrievable—it must be, or the picture couldn’t appear in search results—it’s not searchable or sortable or facetable or anything else. All an application can do with a picture is retrieve it.

As is evident from the figure, the last two fields stand apart from the others. The first one, Stock, contains a count of how many of this particular shoe are in the warehouse, while the second, Promotion, is a Boolean that indicates whether there's a special promotion running for this shoe. But neither one has any attributes set—an application can't even retrieve these values.

What's the point of having data in a search index that can't be returned in search results? The answer is that these fields are used to control the order of those results. Along with a schema, the creator of an index can create a scoring profile that affects the order in which search results are returned. In this example, the scoring profile uses the values of Stock and Boolean to manipulate this order in a way that makes business sense.

It's also worth pointing out that an index can contain fields with geo-spatial types. This allows searches based on location, such as finding a gas station within 10 kilometers of my current location. Geo-spatial types can also be used to support a map-based search interface, letting a user draw a polygon on the map and return, say, information about all of the bookstores within that polygon.

Finally, although all of the text in Figure 6 is in English, Azure Search supports more than 50 languages. You can create indexes that use these languages, and your users can issue search queries in them as well. This multi-language support relies on the same technology for natural language processing that's used by Office and Bing, and it includes a range of capabilities. For example, the technology supports *stemming*, which means that a search keyword such as "earning" can also return results for "earned", "earns", and other variations of the root word "earn". Azure Search's multi-language support also understands plurals, so a search for "mouse" can return results for "mice", too. All of this can be done in any of the languages Azure Search supports, a feature that embodies many person-years of effort.

Populating an Index

Once an index exists, it needs to contain data before an application can issue searches against it. An application can populate an index by making the appropriate calls to the RESTful interface that Azure Search exposes. It's also possible to use an *indexer*, described later, to initially populate an index. However it's done, Figure 7 shows a small part of the data that might be stored in an index with the schema just described.

Category	Brand	Style	Color	Price	Picture	Stock	Promotion
Sneakers	Contoso	HiTops	White, Black	$29.95	http://...	194	True
High Heels	Contoso	Pumps	Red	$129.95	http://...	385	True
High Heels	Contoso	Stilettos	Red, Black	$350.00	http://...	23	True
Boots	Contoso	Beatle	White, Black	$134.79	http://...	100	True
High Heels	Fabrikam	Lipstick Heels	Pink	$489.95	http://...	10	False
High Heels	Fabrikam	Flower	Red	$59.99	http://...	198	False
Tuxedo	Fabrikam	Black	Black	$500.00	http://...	34	False
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Figure 7: An index for an online shoe retailer might include the information shown above.

For the most part, this data looks as you'd expect. Each entry describes a particular kind of shoe, including a link to a picture of that shoe. Two things are worth pointing out about the rightmost fields, however. Notice the large number of Contoso Pumps in stock (shown in the table's second row). The buyer for this store must have really liked these shoes. Notice also that the store is running a promotion for Contoso brand shoes—all of Contoso's records have Promotion set to true. Both of these things can be used to affect the ordering of search results, as described later.

Issuing Searches

Once an index has been created and populated with data, users can begin issuing searches. Figure 8 shows how the search process looks.

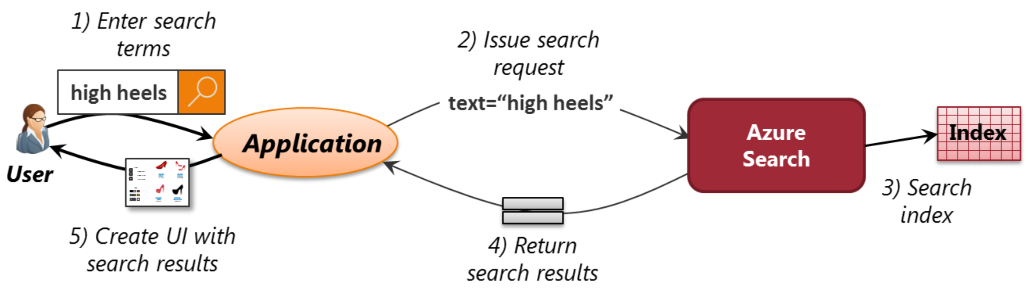
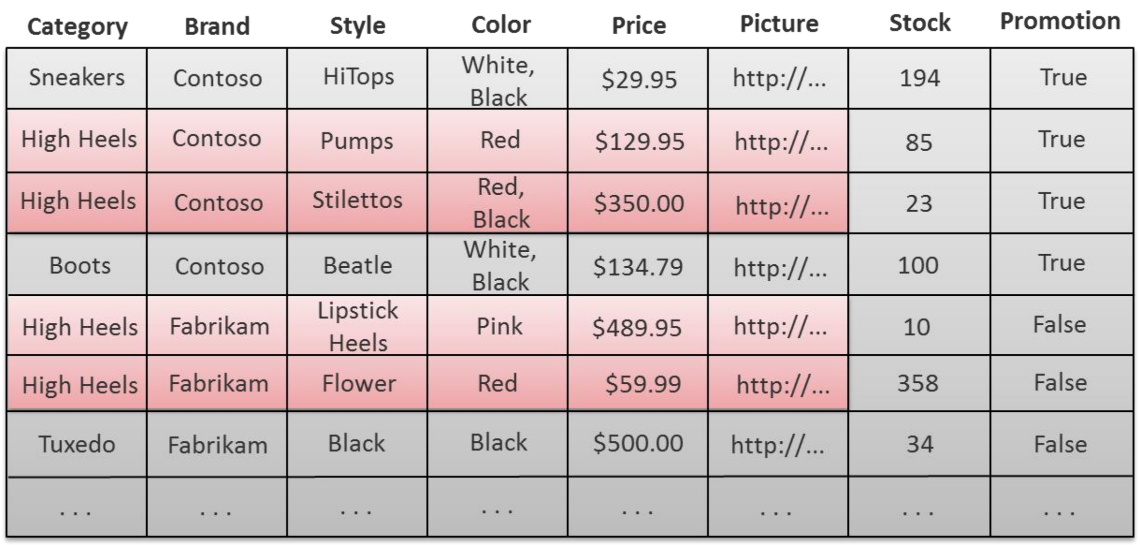


Figure 8: Search terms entered by a user are looked up in an index, with the results sent back and displayed.

A search begins with the user typing her desired search terms, "high heels", into a search box (step 1). As the user is typing, the application can make regular calls to Azure Search to retrieve suggestions to show the user. The application then issues a search request containing the text the user wishes to search for (step 2). Azure Search searches the index for this text (step 3) and returns the search results expressed in JSON (step 4). The application then creates whatever UI it likes to display those results (step 5). Figure 9 shows what data would be returned by a search for “high heels” on our example index.



Category	Brand	Style	Color	Price	Picture	Stock	Promotion
Sneakers	Contoso	HiTops	White, Black	$29.95	http://...	194	True
High Heels	Contoso	Pumps	Red	$129.95	http://...	85	True
High Heels	Contoso	Stilettos	Red, Black	$350.00	http://...	23	True
Boots	Contoso	Beatle	White, Black	$134.79	http://...	100	True
High Heels	Fabrikam	Lipstick Heels	Pink	$489.95	http://...	10	False
High Heels	Fabrikam	Flower	Red	$59.99	http://...	358	False
Tuxedo	Fabrikam	Black	Black	$500.00	http://...	34	False
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Figure 9: By default, a search for “high heels” will return the values shown in red.

The user’s search term, “high heels” matches the Category value for several rows in the index. Because Category was marked as filterable, the search results can be constrained to contain only these rows (shown in red in Figure 9.) Notice that even in the selected rows, however, the values for Stock and Promotion aren’t returned. Neither one is marked as retrievable in this index’s schema, so they’ll never appear in search results.

To make it easier to display text-based results with highlighted search terms, as shown back in Figure 3, Azure Search wraps those terms in an HTML <em> tag in the JSON results it sends back. With these search results, for example, all occurrences of the phrase “high heels” would be returned like this: <em>high heels</em>.

Other aspects of Azure Search, such as the scoring profile and facets, can also be used to determine how the results look. To help make this clear, Figure 10 once again illustrates the simple graphical search results shown earlier, adding some annotations.

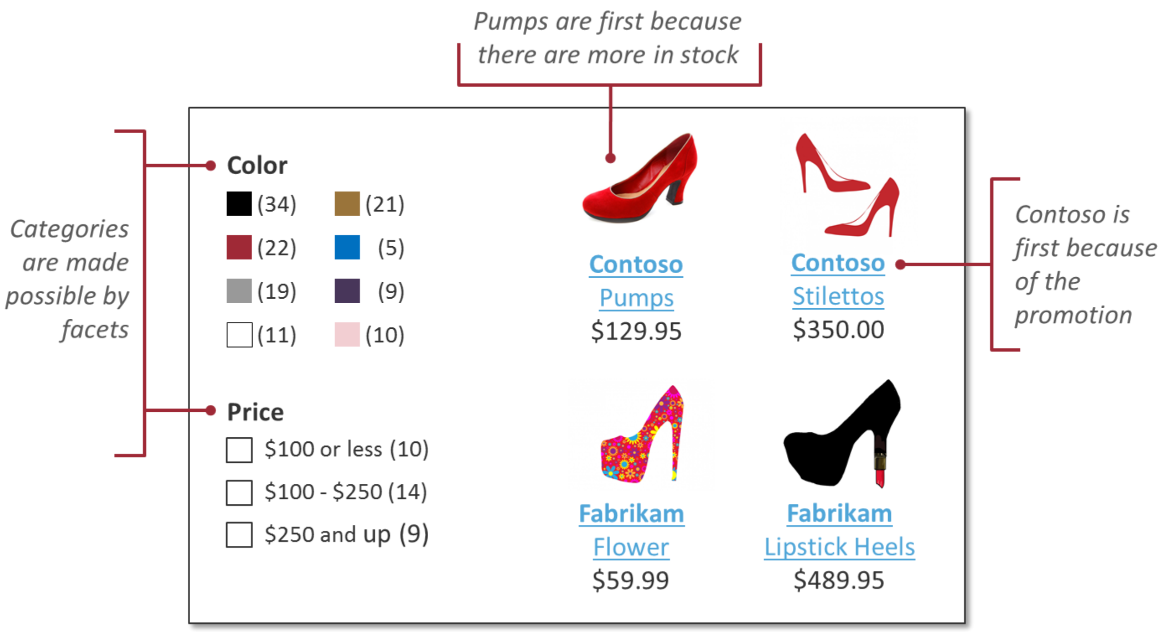

Pumps are first because there are more in stock
Categories are made possible by facets
Contoso is first because of the promotion
Color
■ (34)
■ (21)
■ (22)
■ (5)
■ (19)
■ (9)
□ (11)
■ (10)
Price
□ $100 or less (10)
□ $100 - $250 (14)
□ $250 and up (9)
Contoso Pumps $129.95
Contoso Stilettos $350.00
Fabrikam Flower $59.99
Fabrikam Lipstick Heels $489.95

Figure 10: The way results are displayed can be determined by the data in the index, attribute settings in the schema, the scoring profile, and more.

Remember that the goal of Azure Search is to provide the information applications need to create an effective search UI. The example UI shown here was created using the values shown in red in Figure 9. The category, brand, style, and price of the shoes reflect values in the index data shown earlier, while the pictures come from the JPEGs referenced by the URL in each shoe's Picture field.

The categories on the left, offering color choices and price ranges, are straightforward for the application to create because the Color and Price field are both facetable. Setting this option lets the application make a simple query to get the count for each color or the number of shoes in each price range.

The order in which results are displayed also relies on the index data. Recall that the Promotion field was set to true for all Contoso shoes. This is why Contoso's products show up first in the results, above all shoes from Fabrikam. And within these results, Contoso Pumps appear in the upper left, the prime position. This is because there are so many of them in the warehouse—the company wants to get rid of them. Both of these things are determined by the scoring profile defined for this index.

Updating Indexes

Data changes, and so search indexes must be updated to reflect those changes. Internet search engines such as Bing and Google rely largely on crawler software to do this. Their crawlers run continuously, updating search indexes as they find changes.

Similarly, Azure Search provides a way to automatically have changes in operational data reflected in an index. Doing this relies on *indexers*, as shown in Figure 11.

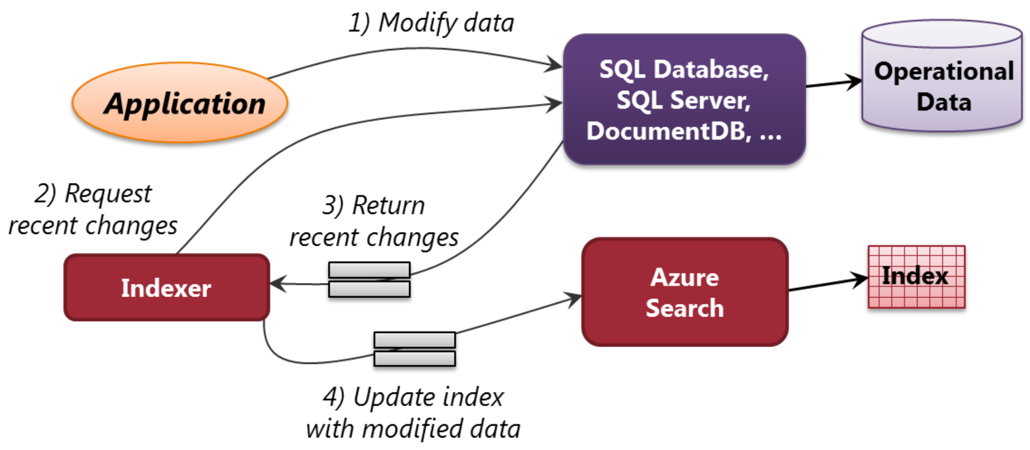

1) Modify data
Application
SQL Database, SQL Server, DocumentDB, ...
Operational Data
2) Request recent changes
3) Return recent changes
Indexer
Azure Search
Index
4) Update index with modified data

Figure 11: Search indexes can be automatically updated to reflect changes in the data being searched.

An application modifies data in its operational store (step 1). An indexer can periodically request recent changes from that store (step 2), get that modified data (step 3), and use it to update the relevant search index (step 4). In its initial release, Azure Search provides indexers for three operational databases on Azure: SQL Database, SQL Server running in an Azure virtual machine, and DocumentDB. Others are planned, so expect this list to grow over time.

Using indexers requires just configuration—there's no need to write any code. For example, a developer can set how frequently an indexer should run, and thus control how up-to-date the search data will be. In its first release, Azure Search allows an indexer to run no more than once every five minutes.

Updating an index every five minutes is fine for some circumstances, but not for everything. For example, think about a user-generated content site, one of the key scenarios that Azure Search addresses. A popular site will have many updates, and each user will expect to see his or her update show up immediately in search results. In this situation, the site's creators might choose to have the application itself update its operational database and its search index at the same time, keeping them constantly in sync. Using the indexers that Microsoft provides isn't required.

Pricing and Scaling

To make it easier to get started with Azure Search, Microsoft provides a free offering. Developers can use this while they're building an application or just to kick the technology's tires. This part of the service is fairly constrained, however, and so it's not meant for more than this. For production use, organizations should choose Azure Search's standard tier.

With this option, a customer buys some number of search units (SUs). Each SU offers a fixed amount of storage, a queries-per-second target, and more. The intent is to let an application pay for exactly the amount of capacity it needs. And because it's possible to add and remove SUs dynamically, an application can scale its search function as needed. If an online shoe store has a big sale, for example, it can purchase a larger number of SUs while the sale is running to handle the increased load, then shut them down—and stop paying—when the sale ends.

Conclusion

How many applications today provide search as a standard part of their UI? The answer is clear: not enough. Given how much users like search, along with their increasing expectations for search support, why doesn't every new application offer this service? The main reason, perhaps, is that search has been hard to implement. Standing up your own search service, then managing it over time, was a high bar.

The goal of Azure Search is to lower this bar. By providing a managed service, it makes life significantly simpler for development teams that want to add search. By running in Azure, it makes this service accessible to all kinds of applications, whether they run in the cloud or on-premises.

The time when search was a nice-to-have feature is likely drawing to a close. Expect to see search, that white box with a magnifying glass, become a mainstream part of new applications.

About the Author

David Chappell is Principal of Chappell & Associates (http://www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.