This is .NET Core MVC app that uses free available version of “StackOverflow.com” database and searches StackOverflow posts for any text entered by user.

This project is created to explore FULL TEXT search feature in SQL server and use of Dapper in .NET Core MVC App.

This is a responsive website and works well on mobile as well.

Follow steps below to explore and use this application –

1. (1A) Download and setup a copy of the StackOverflow.com database locally. This SMALL (10GB) database contains data of posts, users, votes, badges, etc.

<https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/>

Click [here](#DM_Details) for Data Model details.

Refer this external [link](https://stackoverflow.com/questions/4037908/sql-server-importing-database-from-mdf#:~:text=In%20the%20'Object%20Explorer'%20window,database%20and%20you%20are%20done.) for information on how to load MDF file.

OR

(1B) If you do not want to copy entire 10 GB database, you can create a smaller subset of it for quick testing purpose. Follow steps [here](#SmallSubsetDB_Steps).

**Note** - *Step 2 is not required if you follow this approach. However, do read the details on indexes and stored procedures used here.*

1. Add DB indexes and stored procedures in database. See detailed steps [here](#SP_Details). (This step is not needed if you have followed step 1B.)
2. Clone the GitHub repository .
3. Update SQL connection string for *StackDbConnection* in *appsettings.json*
4. Build and run the application.
5. Click [here](#App_Details_SS) for more details on solution and application screenshots.
6. You can see the demo [here](https://stackknowledgebase.azurewebsites.net/). Website deployed on Azure with smaller version of above database (Copied data related to first 10,000 posts only).

**Steps to Create Small Subset of DB**

1. Refer to “[OneTimeSqlsForSmallDbSubset](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/tree/master/OneTimeSqlForSmallDbSubset)”
2. First run [OneTimeLoadDatabaseSchema.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/OneTimeLoadDatabaseSchema.sql). It creates database tables, indexes, and stored procedures.
3. Then run following queries that add data to the tables –
4. [dbo.Badges.Table\_1.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Badges.Table_1.sql)
5. [dbo.Badges.Table\_2.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Badges.Table_2.sql)
6. [dbo.Comments.Table.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Comments.Table.sql)
7. [dbo.LinkTypes.Table.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.LinkTypes.Table.sql)
8. [dbo.PostLinks.Table.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.PostLinks.Table.sql)
9. [dbo.Posts.Table.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Posts.Table.sql)
10. [dbo.PostTypes.Table.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.PostTypes.Table.sql)
11. [dbo.Users.Table\_1.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Users.Table_1.sql)
12. [dbo.Users.Table\_2.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Users.Table_2.sql)
13. [dbo.Users.Table\_3.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Users.Table_3.sql)
14. [dbo.Users.Table\_4.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Users.Table_4.sql)
15. [dbo.Users.Table\_5.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Users.Table_5.sql)
16. [dbo.Users.Table\_6.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Users.Table_6.sql)
17. [dbo.Votes.Table.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.Votes.Table.sql)
18. [dbo.VoteTypes.Table.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/dbo.VoteTypes.Table.sql)
19. Finally, run [RebuildAllIndexesDB.sql](https://github.com/nilesh-ingulkar/StackKnowledgeBaseApp/blob/master/OneTimeSqlForSmallDbSubset/RebuildAllIndexesDB.sql) to rebuild all the indexes.
20. This creates a smaller subset of database with 10,000 posts in Posts table.

[Back to high level steps](#_top)

**Add DB Indexes and Stored Procedures**

1. Run following queries in database for performance improvement.

CREATE FULLTEXT CATALOG ftPostsCatalog AS DEFAULT;

CREATE FULLTEXT INDEX ON dbo.Posts(Body LANGUAGE 1033, Title LANGUAGE 1033, Tags LANGUAGE 1033) KEY INDEX PK\_Posts\_\_Id ON ftPostsCatalog;

CREATE INDEX IDX\_Votes\_\_PostId ON dbo.Votes(PostId);

CREATE INDEX IDX\_Posts\_\_ParentId ON dbo.Posts(ParentId);

CREATE INDEX IDX\_Badges\_\_UserId ON dbo.Badges(UserId);

Few points to note:

* If you do not have full text feature installed in SQL Server, it must be installed. SQL server allows to create full text catalog but gives “*Full-Text Search is not installed, or a full-text component cannot be loaded*” error if the feature is not installed. Refer this external [link](https://stackoverflow.com/questions/47437742/full-text-search-is-not-installed-or-a-full-text-component-cannot-be-loaded) about this error and instructions on how to install full text feature.
* Make sure to create CATALOG before creating index
* Make sure to use Primary Index **Name** (not primary key) while creating full-text index

1. Create stored procedure to get matching posts based on search text

CREATE PROCEDURE dbo.spGetPostsBySearchText

@SearchText NVARCHAR(200),

@OffsetCount INT,

@SelectCount INT

AS

BEGIN

SET NOCOUNT ON

SELECT FT\_TBL.RANK as SearchRank,

pa.Id,

pa.Title,

pa.Body,

(SELECT count(\*) FROM Votes WHERE PostId = pa.Id) as TotalVotes,

(SELECT count(\*) FROM Posts pb WHERE pb.ParentId = pa.Id AND pb.PostTypeId = 2) as TotalAnswers,

pa.OwnerUserId,

usr.DisplayName,

usr.Reputation,

(SELECT CONVERT(nvarchar(max), b.Name + ',') FROM Badges b

WHERE b.UserId = pa.OwnerUserId ORDER BY b.Date desc

FOR XML PATH('') ) as UserBadges

FROM Posts AS pa

INNER JOIN Users AS usr ON pa.OwnerUserId = usr.Id

INNER JOIN FREETEXTTABLE (Posts, (Body, Title, Tags), @SEARCHTEXT) AS FT\_TBL ON pa.Id = FT\_TBL.[KEY]

WHERE pa.PostTypeId = 1

ORDER BY FT\_TBL.RANK DESC

OFFSET @OffsetCount ROWS FETCH NEXT @SelectCount ROWS ONLY;

END;

* This procedure extracts all the fields required to show on search posts panel.
* It takes 3 arguments – search string, offset and how may records to select.
* It sorts data on RANK.
* FREETEXTTABLE creates a weighted rank based on how strongly the search string matches.

**Reason for using FULLTEXT index** –

* User can put any values in search field. For ex. If user puts “C# .NET sql” in search bar it should search on all 3 keywords and find posts with maximum matches to find most relevant posts for user.
* FREETEXT index works best for all that. Also, it is not case-sensitive.
* It searches for pattern in 3 columns Body, Title, Tags.
* Query performance is very good. Mostly returned results in less than 20 seconds during my testing.

**Reason for secondary indexes on *dbo.Votes(PostId), dbo.Posts(ParentId), dbo.Badges(UserId)*** –

* These given columns are used in stored procedure joins. Their primary key is indexed but search is performed based on other fields as well. So having these secondary indexes improves performance of stored procedure.

1. Create stored procedure to get post count for search text

CREATE PROCEDURE dbo.spGetPostCountBySearchText

@SearchText NVARCHAR(200)

AS

BEGIN

SET NOCOUNT ON

SELECT count(\*) as TotalCount

FROM Posts AS pa

INNER JOIN Users AS usr ON pa.OwnerUserId = usr.Id

INNER JOIN FREETEXTTABLE (Posts, (Body, Title, Tags), @SEARCHTEXT) AS FT\_TBL ON pa.Id = FT\_TBL.[KEY]

WHERE pa.PostTypeId = 1;

END;

It takes 1 argument – search string and returns the total count of posts matching with “search text”

1. Create stored procedure to get post details from post id

CREATE PROCEDURE dbo.spGetPostDetailsText

@PostId INT

AS

BEGIN

SET NOCOUNT ON

SELECT a.Id,

a.PostTypeId,

a.ParentId,

a.OwnerUserId,

a.Title,

a.Body,

b.DisplayName,

a.CreationDate

FROM Posts a,

Users b

WHERE (a.Id = @PostId OR a.ParentId = @PostId)

AND a.OwnerUserId = b.Id

ORDER BY a.PostTypeId, a.Id;

END;

It takes 1 argument – post id.

[Back to high level steps](#_top)

**Data Model Details**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Posts** | | | Id | 1 | | AcceptedAnswerId | 2 | | AnswerCount | 3 | | Body | 4 | | ClosedDate | 5 | | CommentCount | 6 | | CommunityOwnedDate | 7 | | CreationDate | 8 | | FavoriteCount | 9 | | LastActivityDate | 10 | | LastEditDate | 11 | | LastEditorDisplayName | 12 | | LastEditorUserId | 13 | | OwnerUserId | 14 | | ParentId | 15 | | PostTypeId | 16 | | Score | 17 | | Tags | 18 | | Title | 19 | | ViewCount | 20 | | |  |  | | --- | --- | | **PostTypes** | | | Id | 1 | | Type | 2 |  |  |  | | --- | --- | | **Values** | | | **Id** | **Type** | | 1 | Question | | 2 | Answer | | 3 | Wiki | | 4 | TagWikiExerpt | | 5 | TagWiki | | 6 | ModeratorNomination | | 7 | WikiPlaceholder | | 8 | PrivilegeWiki | | |  |  | | --- | --- | | **Users** | | | Id | 1 | | AboutMe | 2 | | Age | 3 | | CreationDate | 4 | | DisplayName | 5 | | DownVotes | 6 | | EmailHash | 7 | | LastAccessDate | 8 | | Location | 9 | | Reputation | 10 | | UpVotes | 11 | | Views | 12 | | WebsiteUrl | 13 | | AccountId | 14 | | |  |  | | --- | --- | | **VoteTypes** | | | Id | 1 | | Name | 2 |  |  |  | | --- | --- | | **Values** | | | **Id** | **Name** | | 1 | AcceptedByOriginator | | 2 | UpMod | | 3 | DownMod | | 4 | Offensive | | 5 | Favorite | | 6 | Close | | 7 | Reopen | | 8 | BountyStart | | 9 | BountyClose | | 10 | Deletion | | 11 | Undeletion | | 12 | Spam | | 13 | InformModerator | | 15 | ModeratorReview | | 16 | ApproveEditSuggestion | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Badges** | | | Id | 1 | | Name | 2 | | UserId | 3 | | Date | 4 | | |  |  | | --- | --- | | **Comments** | | | Id | 1 | | CreationDate | 2 | | PostId | 3 | | Score | 4 | | Text | 5 | | UserId | 6 | | |  |  | | --- | --- | | **LinkTypes** | | | Id | 1 | | Type | 2 |  |  |  | | --- | --- | | **Values** | | | **Id** | **Type** | | 1 | Linked | | 3 | Duplicate | | |  |  | | --- | --- | | **PostLinks** | | | Id | 1 | | CreationDate | 2 | | PostId | 3 | | RelatedPostId | 4 | | LinkTypeId | 5 | |

[Back to high level steps](#_top)

**Solution and App Screenshots**

* Solution has 2 projects.
* **StackKnowledgeBase.DataAccess** – This gets data from database. Used Dapper for data access.
* **StackKnowledgeBase.WebUI** - .NET Core 6.0 MVC Application

|  |
| --- |
| Home Page |
| Adding Search Text |
| Click on Search button (zoom level set to 67% to capture all data)   1. Gets the search results (10 records) 2. Shows fetched data. 3. Shows total questions, total pages, and search text 4. Shows page numbers to select. Shows on top and bottom, so that the user has flexibility. 5. Current page is highlighted blue     Each record detailed view   1. For each record shows – 2. Post Title 3. Maximum of 140 characters from the Description 4. Total # of Votes 5. Total # of Answers 6. The User who asked the question along with their reputation score |
| Any page number is clicked - data on that page is shown.   * Notice changes in URL. * Notice the clicked page number is highlighted. |
| Click on next button “>>” – Shows next 10 pages   * Shows pages from 11-20 * Notice previous button “<<”. This was not available on screen with page 1. * Currently it shows 10 pages to select but it can be easily customized to show more or less pages, as needed. |
| Last Page   * Shows pages from 331-337 * Notice next button “>>” is not available on the screen with last page. |
| Click on previous button “<<” – Shows previous 10 pages   * Shows pages from 321 - 330 |
| Pass negative page number in URL   * Passing 0 or negative page number in URL, brings back to page 1. * Notice “-2” in URL. |
| Pass very high page number in URL   * Passing very high page number in URL that is not available, takes to the last page. * Notice “400000” in URL. |
| Searching different words  Search “Unix”    Search “John” |
| Search word that does not exist  Search “abcdghy”    Search Special Character - “double quotes (””)” |
| Click on search button without putting any search text in textbox   * Shows pop-up for user to fill the data. |
| Clicking on a Post Title opens that post  Post Detail Page |
| Mobile View  Home Page    No search text    Search Results    Search result bottom of the page after scrolling    Post Detail Page |
|  |

[Back to high level steps](#_top)