MONTANA
STATE UNIVERSITY

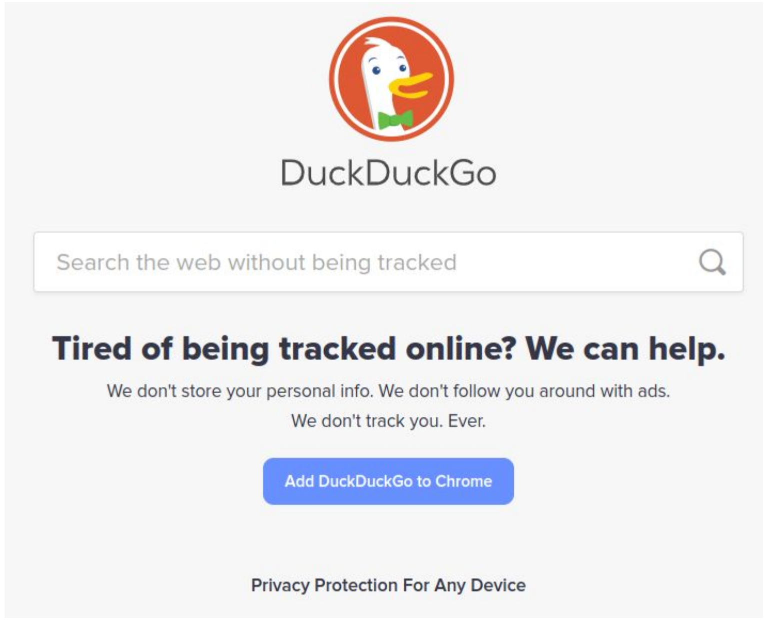
Search

•••

Implementing Search

SEARCH

- One of the most common pieces of functionality to implement in web applications is search
- When you think of search, you no doubt think of duck duck go
  - right?



DuckDuckGo

Search the web without being tracked

Tired of being tracked online? We can help.

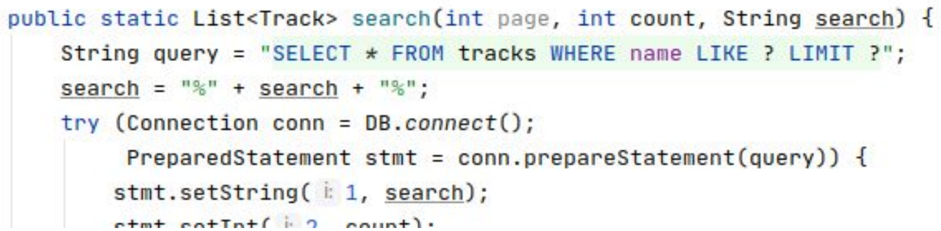
We don't store your personal info. We don't follow you around with ads. We don't track you. Ever.

Add DuckDuckGo to Chrome

Privacy Protection For Any Device

SEARCH

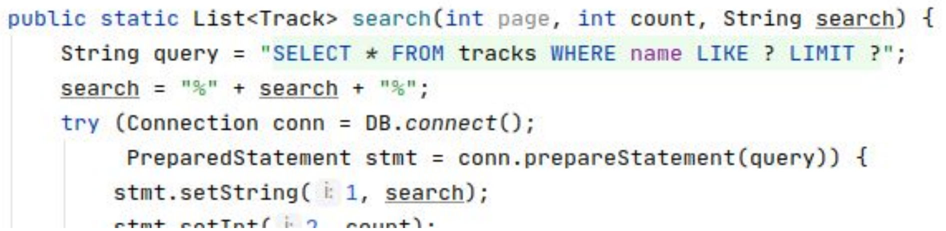
- Most people think of search as textual, as with search engines
- In SQL, general text search is implemented with the LIKE clause
- Recall that wildcards are ‘%’ in SQL, not ‘.*’



```
public static List<Track> search(int page, int count, String search) {
    String query = "SELECT * FROM tracks WHERE name LIKE ? LIMIT ?";
    search = "%" + search + "%";
    try (Connection conn = DB.connect();
          PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString( i: 1, search);
        stmt.setInt( i: 2, count);
```

SEARCH

- Here we enclose the search string with wildcards on either side, thus implementing full text search on the name
- This is **not** efficient
  - There is a reason search engines do not use SQL databases
- However, it is *good enough* for many applications



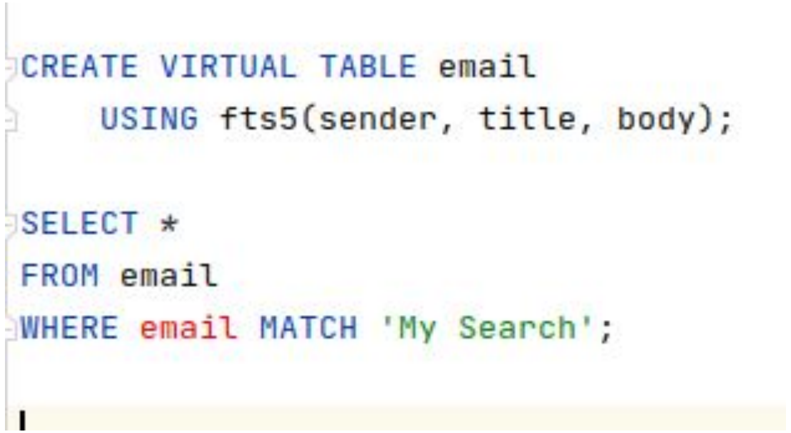
```
public static List<Track> search(int page, int count, String search) {
    String query = "SELECT * FROM tracks WHERE name LIKE ? LIMIT ?";
    search = "%" + search + "%";
    try (Connection conn = DB.connect();
         PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString( 1, search);
        stmt.setInt( 2, count);
```

SEARCH

- If you are interested in *efficient full text search* for large data sets, you would typically use a system designed for that problem
- In Java: Apache Lucene
    - https://lucene.apache.org/core/

SEARCH

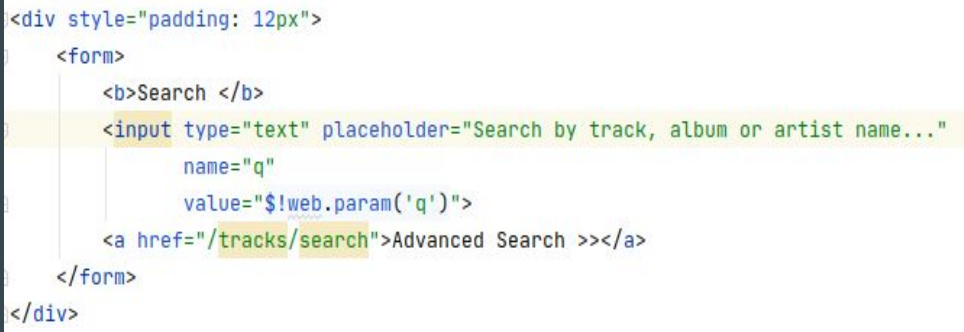
- Most databases now include a text search extension
- SQLite: FTS5 (Full Text Search 5)
  - https://www.sqlite.org/fts5.html
- This is DB specific
  - We will not be using it



```
CREATE VIRTUAL TABLE email
    USING fts5(sender, title, body);

SELECT *
FROM email
WHERE email MATCH 'My Search';
```

SEARCH IMPLEMENTATION

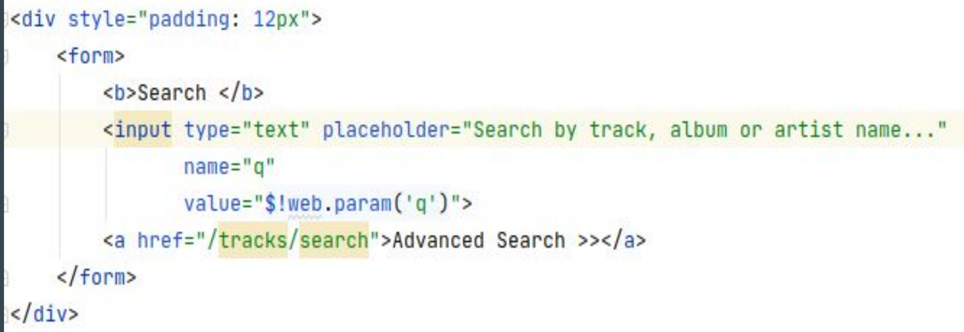
- We are going to use the standard SQL LIKE support to implement search for things like Tracks, Artists, etc.
- This is very much like you would do things in most web applications



```
<div style="padding: 12px">
    <form>
        <b>Search </b>
        <input type="text" placeholder="Search by track, album or artist name..."
               name="q"
               value="$!web.param('q')">
        <a href="/tracks/search">Advanced Search >></a>
    </form>
</div>
```

SEARCH IMPLEMENTATION

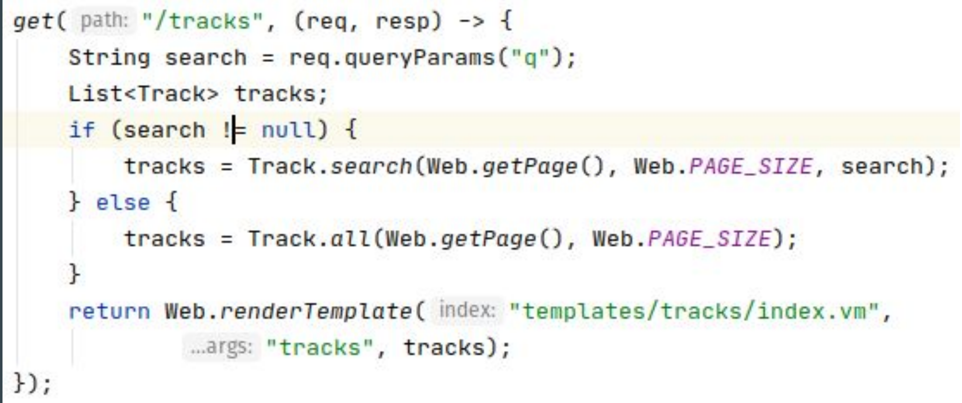
- First we need a search form
  - Note that we do not specify a URL or Action so we will get the default:
    - The current page
    - GET
- When you hit *enter* in this input, it will submit a GET with the parameter ‘q’ (for query)



```
<div style="padding: 12px">
    <form>
        <b>Search </b>
        <input type="text" placeholder="Search by track, album or artist name..."
               name="q"
               value="$!web.param('q')">
        <a href="/tracks/search">Advanced Search >></a>
    </form>
</div>
```

SEARCH IMPLEMENTATION

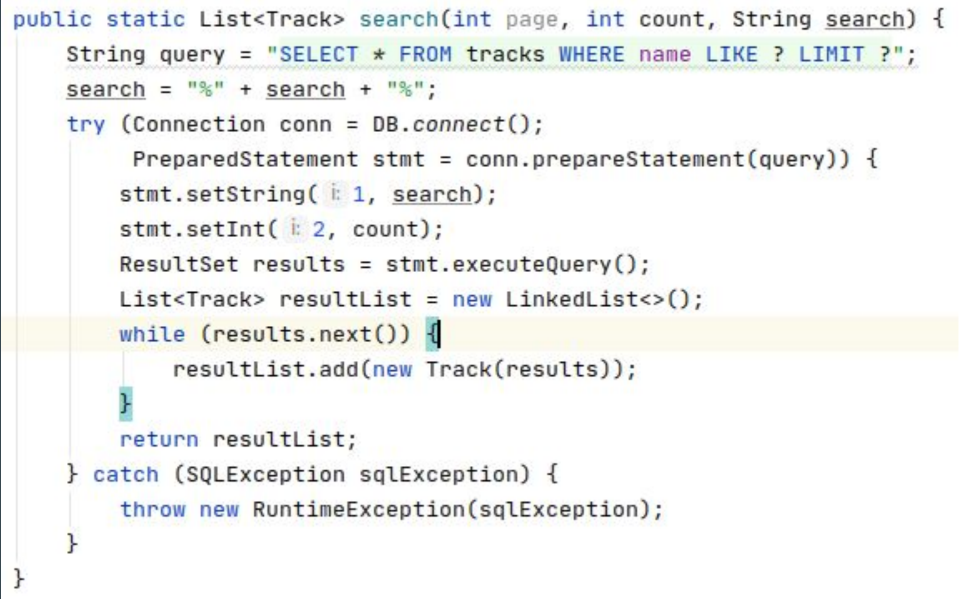
- In the controller, we check for this parameter and, if it is there, call *search()* rather than *all()*



```
get( path: "/tracks", (req, resp) -> {
    String search = req.queryParams("q");
    List<Track> tracks;
    if (search != null) {
        tracks = Track.search(Web.getPage(), Web.PAGE_SIZE, search);
    } else {
        tracks = Track.all(Web.getPage(), Web.PAGE_SIZE);
    }
    return Web.renderTemplate( index: "templates/tracks/index.vm",
            ...args: "tracks", tracks);
});
```

SEARCH IMPLEMENTATION

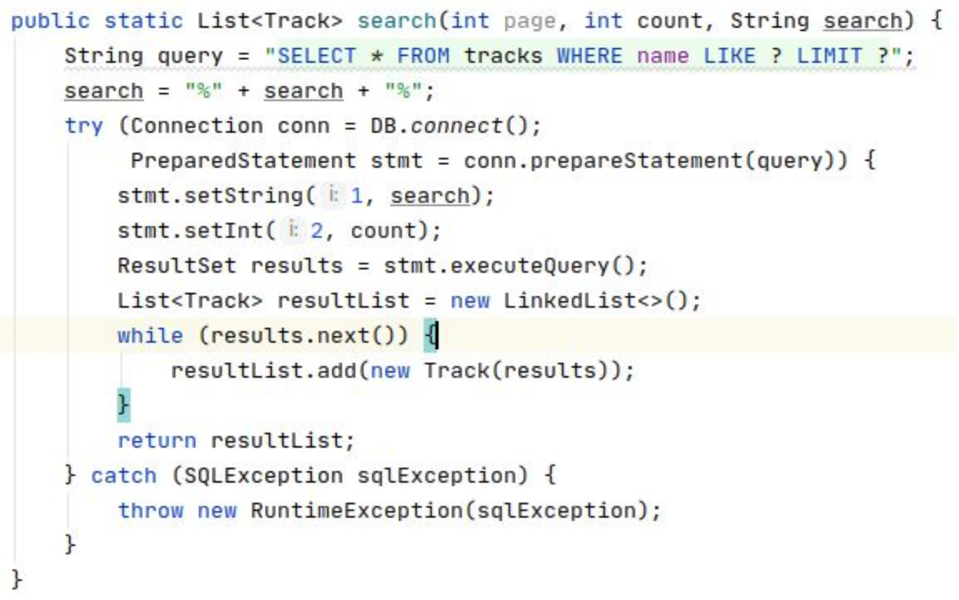
- In the model, we add a method called *search()* that looks a lot like *all()* but takes a search parameter
- IRL you would probably combine these, but let’s leave them separate for clarity



```java
public static List<Track> search(int page, int count, String search) {
    String query = "SELECT * FROM tracks WHERE name LIKE ? LIMIT ?";
    search = "%" + search + "%";
    try (Connection conn = DB.connect();
          PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString( i: 1, search);
        stmt.setInt( i: 2, count);
        ResultSet results = stmt.executeQuery();
        List<Track> resultList = new LinkedList<>();
        while (results.next()) {
            resultList.add(new Track(results));
        }
        return resultList;
    } catch (SQLException sqlException) {
        throw new RuntimeException(sqlException);
    }
}
```

SEARCH IMPLEMENTATION

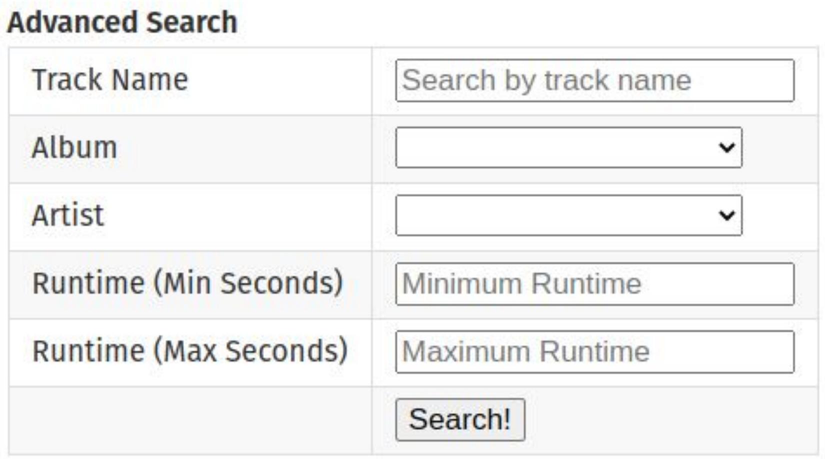
- The search SQL isn't complete: we should be able to search on artist and album too
- That's part of your project to fix



```java
public static List<Track> search(int page, int count, String search) {
    String query = "SELECT * FROM tracks WHERE name LIKE ? LIMIT ?";
    search = "%" + search + "%";
    try (Connection conn = DB.connect();
          PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString( i: 1, search);
        stmt.setInt( i: 2, count);
        ResultSet results = stmt.executeQuery();
        List<Track> resultList = new LinkedList<>();
        while (results.next()) {
            resultList.add(new Track(results));
        }
        return resultList;
    } catch (SQLException sqlException) {
        throw new RuntimeException(sqlException);
    }
}
```

ADVANCED SEARCH

- Not all searches are text searches
- For the project we are going to implement advanced track search functionality
  - Numeric aspects
  - Relational aspects
  - Text aspects

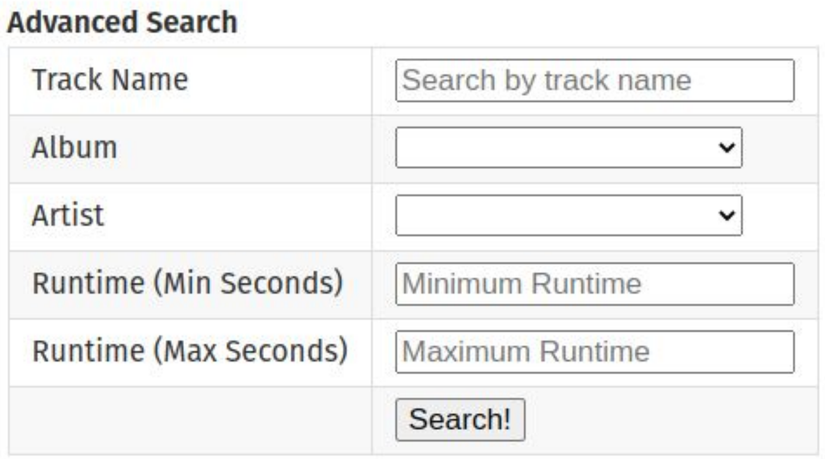


Advanced Search

Track Name	Search by track name
Album	
Artist	
Runtime (Min Seconds)	Minimum Runtime
Runtime (Max Seconds)	Maximum Runtime
	Search!

ADVANCED SEARCH

- This is where databases shine when compared with general text search tools

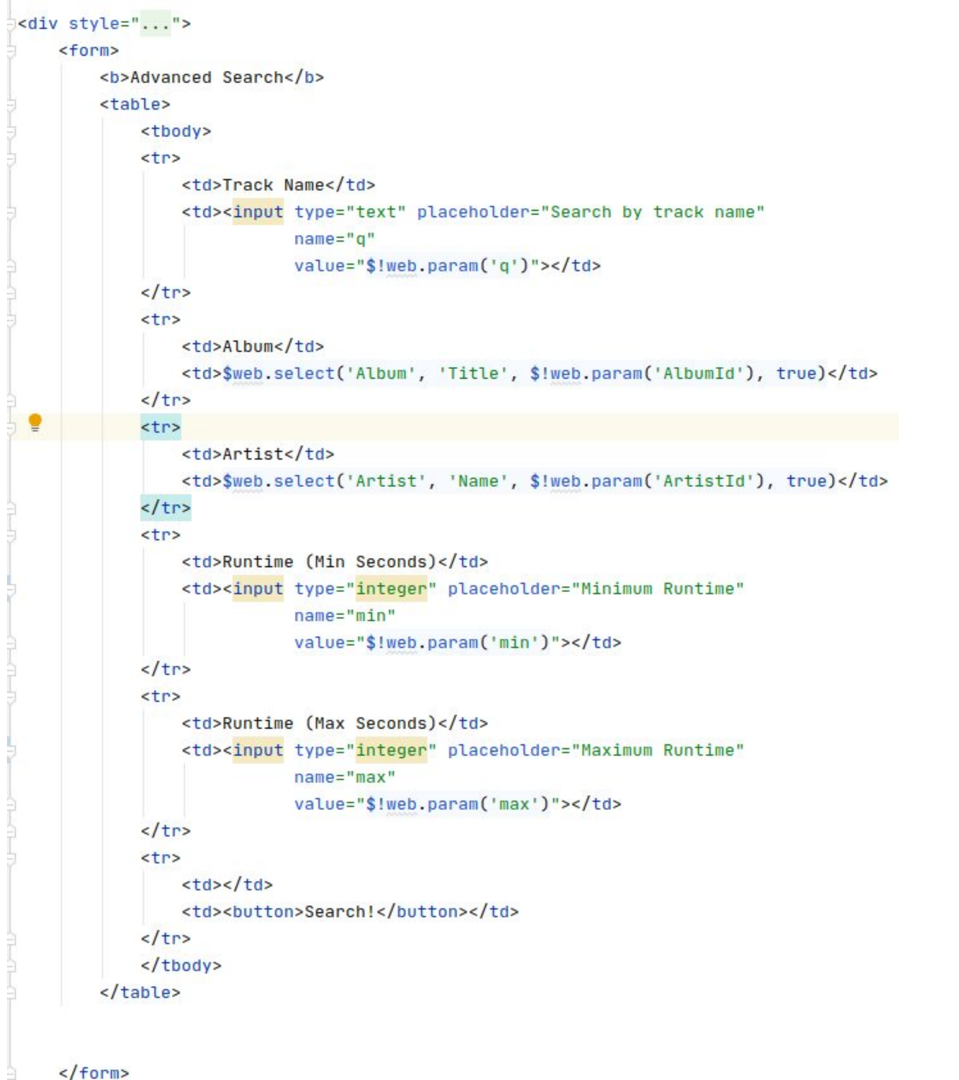


Advanced Search

Track Name	Search by track name
Album	
Artist	
Runtime (Min Seconds)	Minimum Runtime
Runtime (Max Seconds)	Maximum Runtime
	Search!

ADVANCED SEARCH

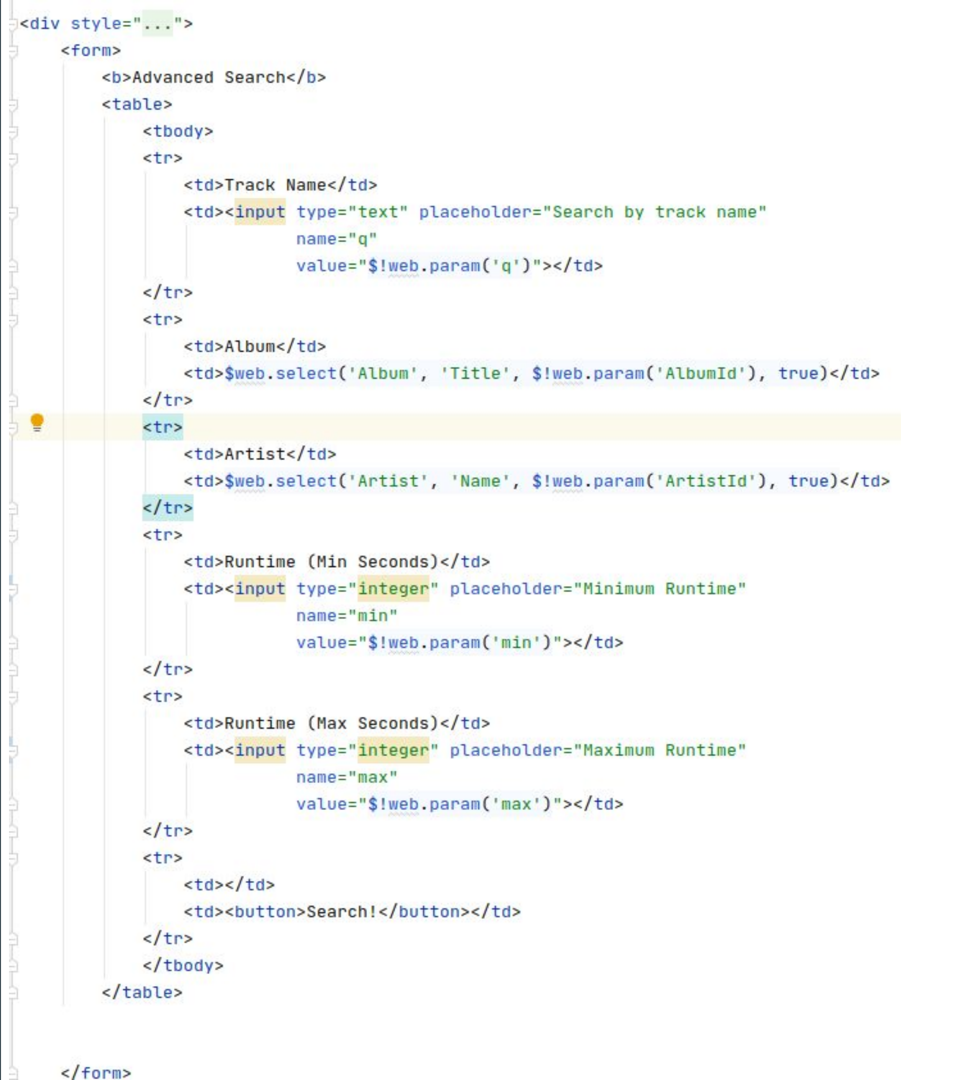
- First, we have an expanded search template
- We have inputs for various fields that are going to make up the search
    - Some text
    - Some drop downs for relational aspects
    - Some numbers



```
<div style="...">
    <form>
        <b>Advanced Search</b>
        <table>
            <tbody>
            <tr>
                <td>Track Name</td>
                <td><input type="text" placeholder="Search by track name"
                           name="q"
                           value="$!web.param('q')"></td>
            </tr>
            <tr>
                <td>Album</td>
                <td>$web.select('Album', 'Title', $!web.param('AlbumId'), true)</td>
            </tr>
            <tr>
                <td>Artist</td>
                <td>$web.select('Artist', 'Name', $!web.param('ArtistId'), true)</td>
            </tr>
            <tr>
                <td>Runtime (Min Seconds)</td>
                <td><input type="integer" placeholder="Minimum Runtime"
                           name="min"
                           value="$!web.param('min')"></td>
            </tr>
            <tr>
                <td>Runtime (Max Seconds)</td>
                <td><input type="integer" placeholder="Maximum Runtime"
                           name="max"
                           value="$!web.param('max')"></td>
            </tr>
            <tr>
                <td></td>
                <td><button>Search!</button></td>
            </tr>
            </tbody>
        </table>


    </form>
```

ADVANCED SEARCH

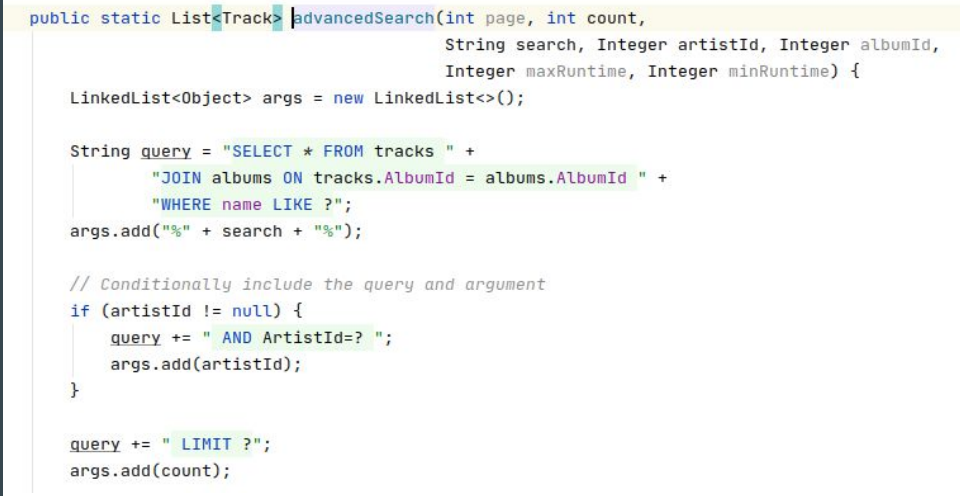
- The controller logic is a bit more involved than the simple search
  - We need to pick out all the values passed in, not just the text search
  - We need to do so in a way that passes *null* to signal “no value”



```
<div style="...">
    <form>
        <b>Advanced Search</b>
        <table>
            <tbody>
            <tr>
                <td>Track Name</td>
                <td><input type="text" placeholder="Search by track name"
                           name="q"
                           value="$!web.param('q')"></td>
            </tr>
            <tr>
                <td>Album</td>
                <td>$web.select('Album', 'Title', $!web.param('AlbumId'), true)</td>
            </tr>
            <tr>
                <td>Artist</td>
                <td>$web.select('Artist', 'Name', $!web.param('ArtistId'), true)</td>
            </tr>
            <tr>
                <td>Runtime (Min Seconds)</td>
                <td><input type="integer" placeholder="Minimum Runtime"
                           name="min"
                           value="$!web.param('min')"></td>
            </tr>
            <tr>
                <td>Runtime (Max Seconds)</td>
                <td><input type="integer" placeholder="Maximum Runtime"
                           name="max"
                           value="$!web.param('max')"></td>
            </tr>
            <tr>
                <td></td>
                <td><button>Search!</button></td>
            </tr>
            </tbody>
        </table>


    </form>
```

ADVANCED SEARCH

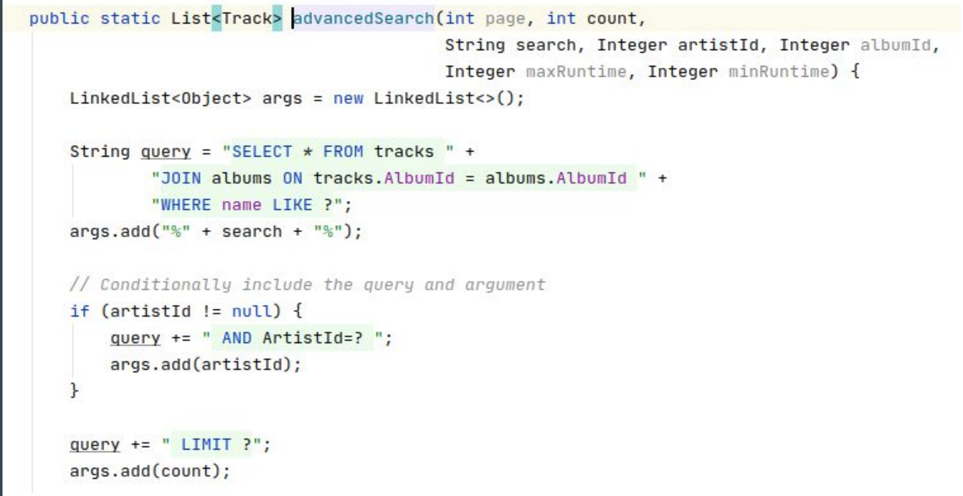
- The model logic is more difficult
- We are constructing a *dynamic* query: the form of the query depends on what the user inputs



```java
public static List<Track> advancedSearch(int page, int count,
                                         String search, Integer artistId, Integer albumId,
                                         Integer maxRuntime, Integer minRuntime) {
    LinkedList<Object> args = new LinkedList<>();

    String query = "SELECT * FROM tracks " +
            "JOIN albums ON tracks.AlbumId = albums.AlbumId " +
            "WHERE name LIKE ?";
    args.add("%" + search + "%");

    // Conditionally include the query and argument
    if (artistId != null) {
        query += " AND ArtistId=? ";
        args.add(artistId);
    }

    query += " LIMIT ?";
    args.add(count);
```

ADVANCED SEARCH

- We must construct the query via string concatenation and keep track of the arguments that will be passed in
- We use a list for the values
- This allows us to use an indexed loop over the variables and ensures that everything works out



```java
public static List<Track> advancedSearch(int page, int count,
                                         String search, Integer artistId, Integer albumId,
                                         Integer maxRuntime, Integer minRuntime) {
    LinkedList<Object> args = new LinkedList<>();

    String query = "SELECT * FROM tracks " +
            "JOIN albums ON tracks.AlbumId = albums.AlbumId " +
            "WHERE name LIKE ?";
    args.add("%" + search + "%");

    // Conditionally include the query and argument
    if (artistId != null) {
        query += " AND ArtistId=? ";
        args.add(artistId);
    }

    query += " LIMIT ?";
    args.add(count);
```

Search Summary

- Text search if a fundamental feature of the web
- “Real” text search uses specialized technology
- We will be using LIKE queries in SQL to implement text search
- Simple text searches are often integrated directly into list views
- With a database, you can also implement more advanced searches using fields with particular data types

MONTANA
STATE UNIVERSITY