Biological Databases - Report

1. Introduction

Purpose and Overview

The scope of this script is to query information about different stable gene IDs from Ensembl, and based on the extracted information, it then queries two additional databases, UniProt and NCBI. Thus, the collected data is used to build three tables in a custom MySQL database, where a combined view of the data is presented and can be extracted as a TSV file.

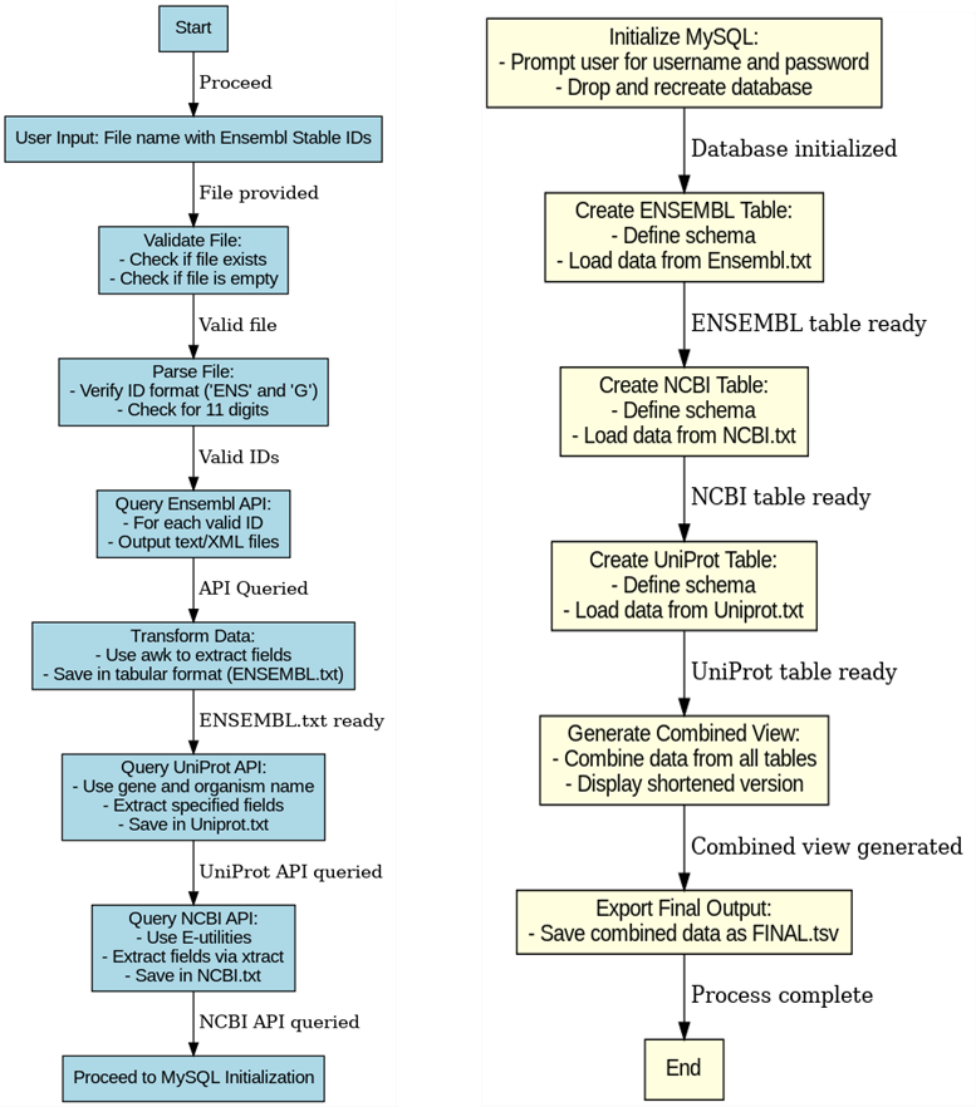

Figure 1: Flow chart of the MySQL and Querying Databases Process

2. Highlights of Important Code

Code Block 1: Script for Validating and Loading Ensembl Stable IDs

```
# We are asking the user if they want to load the script
read -p $'\nWould you like to start the script (YES/NO): ' load_script

# If they input "YES" then we ask them what file they want to load
if [[ ${load_script} == YES ]]
then
    read -p $'\nWhat is the name of the file you want to load: ' sequences_input

    echo -e "\n--------------------------------------------------------------------------
------------------------------"

# If the file is not empty then we proceed to check if the format is correct
    if [ -s ${sequences_input} ]
    then
        echo -e "\n[INFO] Loading Ensembl Stable ID's from file:
${sequences_input}..."

# This while loop checks if the format of the Ensembl ID is correct

# ENS[species prefix][feature type prefix][a unique eleven-digit number]

# but only for Gene feature type because the script only works with genes.


        while read wholeline
        do
            if [[ ${wholeline:0:3} == "ENS" ]]
            then
                if [[ ${wholeline:6:1} == "G" ]]
                then
                    unique=${wholeline:7:11}

                    if [[ ${#unique} == 11 ]]
                    then
                        echo -e "\n[SUCCESS] Valid Ensembl Stable ID found:
${wholeline}"
                        continue
                    else
                        echo -e "\n[FAILURE] Invalid Ensembl Stable ID format,
after Feature Type Prefix: ${wholeline:7:11} Exiting Script...\n"
                        return
                    fi

# The elif statement is required for human stable IDs
                elif [[ ${wholeline:3:1} == "G" ]]
                then
                    unique2=${wholeline:4:11}

                    if [[ ${#unique2} == 11 ]]
                    then
                        echo -e "\n[SUCCESS] Valid Ensembl Stable ID found:
${wholeline}"
                        continue
```

```
                    else
                        echo -e "\n[FAILURE] Invalid Ensembl Stable ID format,
after Feature Type Prefix: ${wholeline:4:11} Exiting Script...\n"
                        return
                    fi
                else
                    echo -e "\n[FAILURE] Invalid Ensembl Stable ID Feature Type
Prefix: ${wholeline} Exiting Script...\n"
                    return
                fi
            else
                echo -e "\n[FAILURE] Invalid Ensembl Stable ID format:
${wholeline} Exiting Script...\n"
                return
            fi
        done < "${sequences_input}"

# If the file is empty a warning is echoed and the script is exited
    else
        echo -e "\n[FAILURE] File ${sequences_input} not found. Exiting the
script...\n"
        return
    fi
fi
```

The first key feature of the script to expand on is showcased in Code Block 1, where the main function is to prompt the user for the name of the file containing the Ensembl stable IDs, please note to include special characters like “\n” in a read -p input statement, a “$” must be included to enable this functionality. Based on that input, the script performs error handling for potential issues with the Ensembl IDs located in the input file.

First, it checks whether the file exists or is empty, as the condition if [-s ${sequences_input}] handles both cases. Next, a while loop goes through each line of the input file, verifying the presents of a correct format, as specified on the Ensembl website (*Ensembl Stable ID Format, n.d.*). The first condition which is being checked, is if the first 3 characters start with ENS and whether the type of prefix indicates a gene (G). Additionally, it confirms that the ID consists of an 11-digit number, and it also includes similar conditional checks for human stable IDs, which differ with a lack of a species element in the stable ID, causing the type of prefix to appear in a different position.

The process relies primarily on the substring parameter expansion, where ${parameter:offset:length} specifies the expected location of ENS and G. Subsequently, by including a “#” at the start of a variable (e.g., ${#unique2}), the script counts the number of characters in the variable rather than reading the actual content. However, this does interfere with substring parameter expansion thus, to address this, the required string is first extracted into a variable and then counted. Finally, if a format problem is encountered, a specific warning is echoed to the terminal based on the location of the error, and the script will exit to inform the user of the mistake, so it can be amended.

Code Block 2: Loop for Querying and Processing Ensembl Data

```
# This is a variable that holds the input file

sequence_file="./${sequences_input}"

# The while loop takes the stable ids from the input file and extracts
```

```
# information for each one of them and then inputs it in the Ensembl.txt

while read sequence_names
do
    echo -e "\n[INFO] Processing sequence ID: ${sequence_names}"
    echo -e "\nQuerying ENSEMBL API for ${sequence_names} data..."

# The wget command requests from the Ensembl rest API information based on the

# stable id's, from the code database of Ensembl and outputted on a text file

# named with a stable id


    wget -q --header="Content-type:text"
"https://rest.ensembl.org/lookup/id/${sequence_names}?db_type=core;format=full" -
O - > ${sequence_names}.txt

# If that text file is not empty, then we remove the information that is not

# needed + the header line with grep and then output the information with tabs on

# one line into Ensembl.txt
    if [ -s ${sequence_names}.txt ]
    then
        cat ${sequence_names}.txt | grep -v "assembly_name" | grep -v "db_type" |
grep -v "object_type" | grep -v "source" | grep -v "version" | grep -v
"logic_name" | grep -v "canonical_transcript" | grep -v "<" | grep -v
"description" | awk 'BEGIN{FS=":";}{ORS="\t";}{print $2}' >> Ensembl.txt

# I am inputting an empty line every instance of the while loop because awk for

# some reason is outputting everything on one line rather at the next line like

# it's supposed too
        echo "" >> Ensembl.txt

        echo -e "\n[SUCCESS] Data extracted for ${sequence_names}. Saving to
Ensembl.txt."
        echo -e "\n----------------------------------------------------------------
---------------------------------"

# The script deletes the previous text files to have a more clean table
        rm -f ./${sequence_names}.txt
        continue

# If the text file is empty, a warning is echoed and the script continued.
    else
        echo -e "\n[WARNING] No data found for sequence ID: ${sequence_names}.
Skipping..."
        echo -e "\n----------------------------------------------------------------
---------------------------------"
        rm -f ./${sequence_names}.txt
        continue
    fi

done < "${sequence_file}"

# After all the stable ids have been queried, we check if our final combined text
```

```
# file has any information


if [ -s Ensembl.txt ]
then
    Ensembl_file=$(cat Ensembl.txt | grep "ENS")

    echo -e "\n============================================================"
    echo -e "ENSEMBL Data Extraction Complete Successfully"
    echo -e "============================================================\n"

# If no data has been found, exit the script since the entire script is based on

# finding data from Ensembl.
else
    echo -e "\n============================================================"
    echo -e "No ENSEMBL Data Extracted. Exiting the Script..."
    echo -e "============================================================\n"
    return
fi
```

Now, that the user's input file has been verified as to hold a valid Ensembl stable ID, the script processes each line of the file in a while loop to query the Ensembl Rest API for the Ensembl core database using the wget command-line utility (*Ensembl Rest API - GET Lookup/Id/:Id*, n.d.), where a variable named ${sequence_names} holds a different stable ID for each line of the users input file (Code Block 2). The output is then saved into a text file named after each Ensembl ID in a text format, as seen in Figure 3. However, since this format is not ideal for use in MySQL, it needs to be transformed into a tabular format. This transformation is achieved using awk to extract the second field from each line and output it into a unified file named ENSEMBL.txt, with the field separator specified as ":" and the output record separator as \t (tab) (Figure 2). This ensures that each field value is output on the same line, separated by a tab, making it easier to load the data into a table later (Figure 2).

A small complication arises in this part of the code, as an empty line needs to be echoed to the end of ENSEMBL.txt each time to ensure proper separation of field values. The script also provides appropriate feedback to the user at each step, while checking if the text file holding the data for each stable ID is empty. Finally, a check is performed on ENSEMBL.txt to ensure it is not empty, as the script depends on information retrieved from the Ensembl core database to proceed.

```
protein_coding  Smug1    103075519   ENSMUSG00000036061  15  mus_musculus  103061717   -1
protein_coding  Itga5    103275190   ENSMUSG00000000555  15  mus_musculus  103252713   -1
protein_coding  Calcoco1  102630613   ENSMUSG00000023055  15  mus_musculus  102615212   -1
protein_coding  Hoxc4    102945278   ENSMUSG00000075394  15  mus_musculus  102927366   1
protein_coding  Hoxc13   102837249   ENSMUSG00000001655  15  mus_musculus  102829538   1
protein_coding  Hoxc5    102925861   ENSMUSG00000022485  15  mus_musculus  102875878   1
protein_coding  Hoxc8    102902543   ENSMUSG00000001657  15  mus_musculus  102899039   1
protein_coding  Hoxc6    102920313   ENSMUSG00000001661  15  mus_musculus  102906689   1
miRNA   Mir615  102923433   ENSMUSG00000076010  15  mus_musculus  102923342   1
protein_coding  Prr13    102371241   ENSMUSG00000023048  15  mus_musculus  102367463   1
```

Figure 2: Screenshot of the ENSEMBL.txt File Content in Tabular Format

```
<html><title>EnsEMBL::REST</title><body><pre>---
assembly_name: GRCm39
biotype: protein_coding
canonical_transcript: ENSMUST00000023128.8
db_type: core
description: integrin alpha 5 (fibronectin receptor alpha) [Source:MGI Symbol;Acc:MGI:96604]
display_name: Itga5
end: 103275190
id: ENSMUSG00000000555
```

Figure 3: Screenshot of the ENSMUSG00000000555.txt File Content in the Text/Xml Format

Coding Block 3: Loop for Querying UniProt API for Specific Data

```
# This while loop reads the Ensembl text file and assigns a variable to each

# field

while read biotype display_name end id seq_region_name species start strand
do
    echo -e "\n[INFO] Processing GENE ID: ${display_name}"
    echo -e "\nQuerying Uniprot API for ${display_name} data..."

# This variable holds the response of the Uniprot Rest Api based on the query

# specified and the fields we required outputted
    Uniprot_responce=$(curl -s -H "Accept: text/plain; format=tsv"
"https://rest.uniprot.org/uniprotkb/search?query=(reviewed:true)%20AND%20(gene:${
display_name})%20AND%20(organism_name:${species})&fields=accession%2Cgene_primary
%2Corganism_name%2Ccc_function%2Cprotein_name%2Csequence" | grep -v "Entry")

# This variable holds the Http response of the Uniprot Rest Api based on the

# query and fields required outputted
    server_response=$(curl -s -o /dev/null -w "%{http_code}" -H "Accept:
text/plain;
format=tsv""https://rest.uniprot.org/uniprotkb/search?query=(reviewed:true)%20AND
%20(gene:${display_name})%20AND%20(organism_name:${species})&fields=accession%2Cg
ene_primary%2Corganism_name%2Ccc_function%2Cprotein_name%2Csequence")

# This if statement checks if the server response is "200" meaning working fine

# and if true we move to the next if statement
    if [[ ${server_response} -eq 200 ]]
    then


# This if statement checks if the Uniprot response is empty, if not then the

# response is outputted into Uniprot.txt, if it is then an echo statement gives a

# warning
        if [[ -n ${Uniprot_responce} ]]
        then
            echo -e "${Uniprot_responce}" >> Uniprot.txt
            echo -e "\n[SUCCESS] Data extracted for ${display_name}. Saving to
Uniprot.txt."
            echo -e "\n-----------------------------------------------------------
---------------------------------------"
        else
            echo -e "\n[WARNING] No data found for GENE ID: ${display_name}"
            echo -e "\n-----------------------------------------------------------
---------------------------------------"
        fi

# This elif statement checks if the server response is "500", If it is then an

# echo statement echoes that there is an internal error
    elif [[ ${server_response} -eq 500 ]]
    then
        echo -e "\n[WARNING] Server internal error. Try re-running the script or
running it at a different time."
```

```
# This else statement handles any other HTTP errors.
    else
        echo -e "\n[WARNING] Received HTTP status code ${server_response}"
    fi

done <<< ${Ensembl_file}
```

The next part of the script to highlight is responsible for querying the UniProt API (Coding Block 3) with the use of the Curl command-line utility this time (*Programmatic Access - Retrieving Entries via Queries*, n.d.). Since UniProt allows data to be downloaded in tabular format, no post-processing of the data is required. This process is nested inside a while loop that iterates through ENSEMBL.txt, where each field is assigned to a variable that can be used in the curl command. The search portion of the curl command specifies that the query is based on the gene and organism name, using variables from ENSEMBL.txt that hold the gene name and organism name for each stable ID. This enables an automatic and unique query for each stable ID without manual intervention.

The fields to be extracted are specified after “fields=”, making it easy to customise based on specific requirements; details about available fields can be found on the UniProt website *(UniProtKB Return Fields*, n.d.), with the output of the pipe stored in a variable named Uniprot_response. Before saving this output to a text file, another variable holds a modified curl command that includes /dev/null and %{http_code}. The former ensures the actual output from curl command is discarded, while the latter retrieves and writes the HTTP response code to the variable. An if statement checks the HTTP response code, and if it returns 200, the Uniprot_response is written to the end of the Uniprot.txt, provided the response is not empty. This process repeats for all lines in ENSEMBL.txt, while providing the user with appropriate feedback, ensuring that any errors are accounted for.

Coding Block 4: Loop for Querying and Processing Data with the Use of NCBI E-Utilities

```
# This while loop reads the Ensembl text file and assigns a variable to each

# field


while read biotype display_name end id seq_region_name species start strand
do
    echo -e "\n[INFO] Processing GENE ID: ${display_name} and ORGANISM:
${species}"


    echo -e "\nQuerying NCBI API for ${display_name} and ${species} data..."

# This code is utilising the E-utilities Api from NCBI to search the Gene

# database based on a specific query, subsequently the data are fetched into an

# xml format and then using xtract the fields that you want to be specified are

# extracted into NCBI.txt in a tabular form.


    esearch -db Gene -query "${display_name}[GENE] AND ${species}[ORGANISM]" <
/dev/null | efetch -format xml | xtract -pattern Entrezgene -element Gene-
ref_locus -block Entrezgene_summary -element Entrezgene_summary >> NCBI.txt

# The use of < /dev/null is necessary because without it the xtract stops
```

```
# interacting with the while loop


    echo -e "\n[SUCCESS] Data extracted for ${display_name} and ${species}.
Saving to NCBI.txt."


    echo -e "\n-----------------------------------------------------------------
------------------------------"

done <<< "${Ensembl_file}"
```

Finally, when it comes to querying a database for information, the use of E-utilities is employed (Coding Block 4). E-utilities is a custom tool developed for querying data from NCBI (Jonathan Kans, 2013). Similar to the UniProt section, a while loop is utilised where the contents of ENSEMBL.txt are stored in variables that are then used to query data. The esearch command is put to use by specifying the desired database and query, which in this case are the gene name and species, based on variables from ENSEMBL.txt.

The next step pipes into efetch, where the output format is set to XML, mainly because it provides more data than a tab-delimited file. The XML data are subsequently piped into xtract, a function included in E-utilities (Jonathan Kans, 2013), which transforms XML data into a tab-delimited format. The fields of choice are extracted, Gene-ref_locus and Entrezgene_summary, which are then piped into the end of NCBI.txt. The main issue encountered was that esearch would not interact properly with the while loop, to resolve this, < /dev/null was included allowing the command to work correctly (*Bash Loop Using Edirect (Esearch|efetch), n.d.)*.

Code Block 5: User Authentication and Custom Database Initialisation

```
# Here we ask the user for what username and password to use

# The "-s" silences the user input thus, protecting the integrity of the database

 read -p $'What is your MySQL username (Hint: s2748062): ' mysql_username

 read -s -p $'\nWhat is your password (Hint: awxcKPiI): ' mysql_password


# Here the code makes sure the database is clear and that it exists

 echo "DROP DATABASE s2748062; CREATE DATABASE s2748062;" | mysql -
u${mysql_username} -p${mysql_password}
```

The conclusion of the script is highlighted by the MySQL custom database creation. As shown in the flowchart (Figure 1), the process begins by prompting the user for their username and password, with the database name already predetermined (Code Block 5). Since the database is based on the University's student number, and the script runs on the Bioinformatics Server 6, no additional prompts were included for the user to input the port, database name, or host address - features that could easily be added and implemented in the code. The MySQL commands are piped via echo statements, and since the user's password input is silenced, the integrity and security of the credentials are maintained.

Code Block 6: Creation and Population of the ENSEMBL Table

```
echo "DROP TABLE IF EXISTS ENSEMBL; CREATE TABLE ENSEMBL (biotype VARCHAR(20),
display_name VARCHAR(50), end INT, id VARCHAR(50) PRIMARY KEY, seq_region_name
VARCHAR(10), species VARCHAR(50), start INT, strand TINYINT); LOAD DATA LOCAL
INFILE './Ensembl.txt' INTO TABLE ENSEMBL FIELDS TERMINATED BY '\t' LINES
TERMINATED BY '\n';" | mysql s2748062 -u${mysql_username} -p${mysql_password}
```

Code Block 7: Generating a Unified Data View and Exporting Results

```
echo -e "SELECT ENSEMBL.*, SUBSTRING(NCBI.Gene_description, 1, 600) AS
Gene_description, SUBSTRING(UNIPROT.accession, 1, 10) AS accession,
SUBSTRING(UNIPROT.cc_function, 1, 1000) AS cc_function,
SUBSTRING(UNIPROT.Protein_Name, 1, 200) AS Protein_Name,
SUBSTRING(UNIPROT.sequence, 1, 100) AS Short_Sequence FROM ENSEMBL LEFT JOIN NCBI
ON TRIM(ENSEMBL.display_name) = TRIM(NCBI.symbol) LEFT JOIN UNIPROT ON
TRIM(ENSEMBL.display_name) = TRIM(UNIPROT.gene_primary) \G"| mysql s2748062 -
u${mysql_username} -p${mysql_password}

# Outputting the table in a tsv so when imported in excel it looks better than a

# csv

        echo -e "\n--------------------------------------------------------------------
-----------------------------------\n"
        echo -e "SELECT ENSEMBL.*, NCBI.Gene_description, UNIPROT.accession,
UNIPROT.cc_function, UNIPROT.Protein_Name, UNIPROT.sequence FROM ENSEMBL LEFT
JOIN NCBI ON TRIM(ENSEMBL.display_name) = TRIM(NCBI.symbol) LEFT JOIN UNIPROT ON
TRIM(ENSEMBL.display_name) = TRIM(UNIPROT.gene_primary)" | mysql s2748062 -
u${mysql_username} -p${mysql_password} > FINAL.tsv
```

Furthermore, the script creates the necessary database schemas and populates the tables with data extracted from three text files (Code Block 6). These files contain the combined data from queries to the Ensembl, UniProt, and NCBI databases (Supplementary Data 2). Subsequently, the script generates a combined view of the data, including a shortened sequence for better clarity, by performing a join operation across the three tables (Code Block 7). This operation links trimmed and related information from the Ensembl, UniProt, and NCBI tables based on a shared identifier. By removing any hidden spaces before or after the identifiers and ensuring accurate comparisons, all relevant fields are consolidated into a single, unified view. Finally, the unified view is exported as a tab-delimited file, enabling access and analysis using various software, such as Excel (Code Block 7).

As shown in the flowchart (Figure1), the code has been built with generalisation in mind. From the stable IDs being provided as user input, to the querying of different databases based on that input, the script is designed to work with all gene stable IDs. The database creation is also flexible, as it can be modified based on user input. The only hardcoded elements are the fields retrieved from each database and the number of columns in each MySQL table. While the code has certain limitations, its foundation is designed to adapt to a variety of different scenarios.

3. Limitations and Challenges

As discussed, when highlighting the core features of the script, multiple challenges were encountered during its development. One such challenge was the validation of the format process, which was addressed through the use of if statements and substring expansion. While this solution works effectively, it currently supports only the expected feature type genes, and does not handle all other types. This could be considered a general limitation of the script, as no information would be extracted for types other than genes currently.

Expanding on that limitation, querying the Ensembl database presented challenges in extracting data from a JSON format into a tab-delimited format without installing additional tools that are not already supported on the bioinformatics server to handle the conversion. To address this, the output was specified as text, which simplified processing. However, this introduced another limitation, the amount of data extracted in the XML format is significantly restricted compared to what could be obtained from JSON. For the purposes of this coursework, it works adequately, but, in a real-life scenario that wouldn't be ideal.

On the other hand, the UniProt query code allows for direct extraction of all possible data in a tabular format, with the only limitation being that the fields must be predetermined and hardcoded before the script runs. This makes the field selection process time-consuming, as the user must reference the UniProt website (*UniProtKB Return Fields*, n.d.).

Similarly, the NCBI query suffers from the same limitation, fields to be extracted must be selected beforehand, as the xtract command converts them into tabular format. This prevents users from dynamically specifying the fields they want. Despite this, the pipe command used for the NCBI query is functional and enables the extraction of all available information through xtract. This flexibility represents a challenge the script wasn't able to overcome when querying Ensembl, as xtract is part of the E-utilities and not readily available for Ensembl specific data processing (Jonathan Kans, 2013).

The last major challenge encountered was the custom database creation, where an analysis of the data and an appropriate definition of the format for each column in each table was needed, as shown in the schema (Supplementary Data 2). One potential limitation of creating the schema in this manner is that the process is not automated. A manual examination of the data is required, along with defining the data types. For instance, if additional data were to be extracted from UniProt, a manual addition of columns to the table schema is needed, making the process time consuming and less flexible for future adjustments.

To add to that, one of the major problems encountered when first using WHERE statements to define a common identifier, was that the UniProt table had nine columns, and certain rows were excluded because no data was found for the Ensembl ID "ENSMUSG00000076010". As a result, when combining the tables, that row of data would not be included, even if information was available from the other databases. This issue was resolved using a LEFT JOIN operation, which ensures that all rows from the primary table are included, even if there is no matching data in the other tables.

Subsequently, another challenge arose, even though the identifiers were the same, hidden spaces before or after the values prevented matches, making the process of combining data impossible. To resolve this, the TRIM statement was used to remove any extraneous spaces, ensuring only actual characters were compared. While this approach is effective, it introduces another requirement, the MySQL command must explicitly specify all the columns that need to be included in the output to avoid duplicate columns appearing.

4. Discussion of Biological Results

The dataset provides valuable insights into the biological roles of several genes, particularly focusing on their functionalities, and protein associations. One notable finding from the final dataset was the biotype of all stable IDs, it revealed that all IDs are protein-coding except for ENSMUSG00000076010, which is classified as miRNA. Additionally, the gene names of all the IDs were successfully identified, with five belonging to the Hoxc gene family and the other five being unique.

Furthermore, the dataset provides precise information on the start and end locations of each gene, as well as their strand orientation, with seven out of ten genes located on strand 1 and the remaining three on strand -1. Interestingly, all the stable IDs share a common chromosomal location on chromosome 15, and they are all derived from the same species, Mus musculus (Mouse).

Based on gene functionality, the HOXC genes (Hoxc13, Hoxc8, Hoxc6, Hoxc5, and Hoxc4) collectively enable DNA-binding activator activity that is RNA polymerase II-specific in some of the HOXC genes. They are primarily localised in the nucleoplasm or nucleus, playing pivotal roles in anterior/posterior pattern specification and are expressed in various structures, including the alimentary system. Moreover, they are orthologous to human HOXC genes showcasing evolutionary importance.

Other genes in the dataset, such as Calcoco1 and Smug1, exhibit distinct functionalities. Calcoco1 facilitates chromatin binding and transcription coactivator activity and is directly associated with chromatin, while Smug1 is predicted to enable DNA N-glycosylase and identical protein binding activity, with localisation in the cytosol, fibrillar centre, and nucleoplasm. Both genes maintain orthology with their human counterparts, CALCOCO1 and SMUG1, respectively.

In contrast, Mir615 represents a unique functional category as a microRNA (miRNA). Unlike the protein-coding genes, Mir615 operates at the post-transcriptional level, regulating gene expression by targeting mRNA stability and translation. It is transcribed as a primary miRNA by RNA polymerase II, processed by Drosha and Dicer enzymes, and incorporated into the RNA-induced silencing complex (RISC) to inhibit or destabilise the target mRNA. This distinctive mechanism sets Mir615 apart from the protein-coding genes.

The dataset also highlights the diverse roles of proteins encoded by the stable IDs identified as protein-coding. For each protein-coding gene, the dataset successfully extracts the corresponding protein name, providing valuable insights into their biological roles. These proteins span a variety of functional categories, emphasising their importance in transcription regulation, enzymatic activity, and cellular signalling.

In conclusion, the sequence of each protein-coding stable ID is also included in the output, providing a valuable resource for further analysis. The overall data extracted from the three databases offers users a comprehensive understanding of the functionality of the genes and their associated proteins. Additionally, supplementary information is provided, which may be deemed important or unimportant depending on the user’s specific needs. Given the vast potential amount of data available, it would be ideal for users to have the ability to selectively extract the information most relevant to their research from the endless sea of information within the databases.

5. Supplementary Data

Supplementary Data 1: API Data Retrieval and MySQL Bash Script

```
#!/usr/bin/bash

echo -e "\n============================================================"
echo -e " Welcome to DatabaseMaker2025!"
echo -e "============================================================\n"

# We are asking the user if they want to load the script

read -p $'\nWould you like to start the script (YES/NO): ' load_script

# If they input "YES" then we ask them what file they want to load

if [[ ${load_script} == YES ]]
then

	read -p $'\nWhat is the name of the file you want to load: '
sequences_input

	echo -e "\n------------------------------------------------------------------
--------------------------------"

# If the file is not empty then we proceed to check if the format is correct

	if [ -s ${sequences_input} ]
	then

		echo -e "\n[INFO] Loading Ensembl Stable ID's from file:
${sequences_input}..."

# This while loop checks if the format of the Ensembl ID is correct

# ENS[species prefix][feature type prefix][a unique eleven-digit number]

# but only for Gene feature type because the script only works with genes.

		while read wholeline
		do

			if [[ ${wholeline:0:3} == "ENS" ]]
			then

				if [[ ${wholeline:6:1} == "G" ]]
				then

					unique=${wholeline:7:11}

					if [[  ${#unique} == 11 ]]
					then
						echo -e "\n[SUCCESS] Valid Ensembl Stable
ID found: ${wholeline}"

						continue
					else
```

```
                                        echo -e "\n[FAILURE] Invalid Ensembl
Stable ID format, after Feature Type Prefix: ${wholeline:7:11} Exiting
Script...\n"
                                        return
                                fi


# The elif statement is required for human stable IDs


                        elif [[ ${wholeline:3:1} == "G" ]]
                        then
                                unique2=${wholeline:4:11}

                                if [[ ${#unique2} == 11 ]]
                                then
                                        echo -e "\n[SUCCESS] Valid Ensembl Stable
ID found: ${wholeline}"
                                        continue
                                else
                                        echo -e "\n[FAILURE] Invalid Ensembl
Stable ID format, after Feature Type Prefix: ${wholeline:4:11} Exiting
Script...\n"
                                        return
                                fi
                        else
                                echo -e "\n[FAILURE] Invalid Ensembl Stable ID
Feature Type Prefix: ${wholeline} Exiting Script...\n"
                                return
                        fi
                else
                        echo -e "\n[FAILURE] Invalid Ensembl Stable ID format:
${wholeline} Exiting Script...\n"
                        return
        fi

        done < "${sequences_input}"

# If the file is empty a warning is echoed and the script is exited

    else
        echo -e "\n[FAILURE] File ${sequences_input} not found. Exiting the
script...\n"

        return
    fi

    echo -e "\n-------------------------------------------------------------------
--------------------------------"
    echo -e "\n[INFO] All sequences have been successfully loaded from
${sequences_input}.\n"
    echo -e "-------------------------------------------------------------------
-----------------------------\n"

# Since we established that the script is going to run, we delete the previous
runs data

    rm -f ./Ensembl.txt # comment -f
```

```
        rm -f ./Uniprot.txt
        rm -f ./NCBI.txt

# This is a variable that holds the input file

        sequence_file="./${sequences_input}"

        touch Ensembl.txt

# This is the start of the Ensembl information extraction process

        echo -e "\n============================================================"
        echo -e "Starting Data Extraction from the ENSEMBL Database"
        echo -e "============================================================\n"

# The while loop takes the stable id's from the input file and extracts

# information for each one of them and then inputs it in the Ensembl.txt

        while read sequence_names
        do

                echo -e "\n[INFO] Processing sequence ID: ${sequence_names}"
                echo -e "\nQuerying ENSEMBL API for ${sequence_names} data..."

# The wget command requests from the Ensembl rest API information based on the

# stable id's, from the code database of Ensembl and outputted on a text file

# named with a stable id

                wget -q --header="Content-type:text"
"https://rest.ensembl.org/lookup/id/${sequence_names}?db_type=core;format=full"
-O - > ${sequence_names}.txt

# If that text file is not empty, then we remove the information that is not

# needed + the header line with grep and then output the information with tabs on

# one line into Ensembl.txt

                if [ -s ${sequence_names}.txt ]
                then
                        cat ${sequence_names}.txt | grep -v "assembly_name" | grep -v
"db_type" | grep  -v "object_type" | grep -v "source" | grep -v "version" | grep
-v "logic_name" | grep -v "canonical_transcript" | grep -v "<" | grep -v
"description" | awk 'BEGIN{FS=":";}{ORS="\t";}{print $2}' >> Ensembl.txt

# I am inputting an empty line every instance of the while loop because awk for

# some reason is outputting everything on one line rather at the next line like

# it's supposed too

                        echo "" >> Ensembl.txt

                        echo -e "\n[SUCCESS] Data extracted for ${sequence_names}.
Saving to Ensembl.txt."
                        echo -e "\n------------------------------------------------
-----------------------------------------"
```

```
# We delete the previous text files to have our folder more clean

                    rm -f ./${sequence_names}.txt
                    continue

# If that text file is empty then we just echo a warning and continue the script

             else
                    echo -e "\n[WARNING] No data found for sequence ID:
${sequence_names}. Skipping..."
                    echo -e "\n-------------------------------------------------
-----------------------------------------"
                    rm -f ./${sequence_names}.txt
                    continue
             fi


       done < "${sequence_file}"

# After all the stable ids have been queried, we check if our final combined text

# file has any information

       if [ -s Ensembl.txt ]
       then
             Ensembl_file=$(cat Ensembl.txt  | grep "ENS")

             echo -e
"\n============================================================="
             echo -e "ENSEMBL Data Extraction Complete Successfully"
             echo -e
"=============================================================\n"

# If no data has been found we exit the script since the whole script is based on

# finding data based on Ensembl

       else
             echo -e
"\n============================================================="
             echo -e "No ENSEMBL Data Extracted. Exiting the Script..."
             echo -e
"=============================================================\n"
             return
       fi

# Here is the start of the Uniprot information extraction process

       echo -e "\n============================================================="
       echo -e " Starting Data Extraction from the UNIPROT Database"
       echo -e "=============================================================\n"

# Creation of the text file where all the data are going to be inserted

       touch Uniprot.txt

# This while loop reads the Ensembl text file and assigns a variable to each
```

```
# field

      while read biotype display_name end id seq_region_name species start strand
      do
            echo -e "\n[INFO] Processing GENE ID: ${display_name}"
            echo -e "\nQuerying Uniprot API for ${display_name} data..."

# This variable holds the response of the Uniprot Rest Api based on the query

# specified and the fields we required outputted

            Uniprot_responce=$(curl -s -H "Accept: text/plain; format=tsv"
"https://rest.uniprot.org/uniprotkb/search?query=(reviewed:true)%20AND%20(gene:${
display_name})%20AND%20(organism_name:${species})&fields=accession%2Cgene_primary
%2Corganism_name%2Ccc_function%2Cprotein_name%2Csequence" | grep -v "Entry")

# This variable holds the Http response of the Uniprot Rest Api based on the

# query and fields required outputted

            server_response=$(curl -s -o /dev/null -w "%{http_code}" -H "Accept:
text/plain; format=tsv"
"https://rest.uniprot.org/uniprotkb/search?query=(reviewed:true)%20AND%20(gene:${
display_name})%20AND%20(organism_name:${species})&fields=accession%2Cgene_primary
%2Corganism_name%2Ccc_function%2Cprotein_name%2Csequence")

# This if statement checks if the server response is "200" meaning working fine

# and if true we move to the next if statement

            if [[ ${server_response} -eq 200 ]]
            then


# This if statement checks if the Uniprot response is empty, if not then the

# response is outputted into Uniprot.txt, if it is then an echo statement gives a

# warning

                  if [[ -n ${Uniprot_responce} ]]
                  then
                        echo -e "${Uniprot_responce}" >> Uniprot.txt
                        echo -e "\n[SUCCESS] Data extracted for
${display_name}. Saving to Uniprot.txt."
                        echo -e "\n----------------------------------------------
--------------------------------------------------"

                  else
                        echo -e "\n[WARNING] No data found for GENE ID:
${display_name}"
                        echo -e "\n----------------------------------------------
--------------------------------------------------"
                  fi

# This elif statement checks if the server response is "500", If it is then an

# echo statement echoes that there is an internal error

            elif [[ ${server_response} -eq 500 ]]
```

```
            then
                  echo "\n[WARNING] Server internal error, try re-running the
script or try re-running it at different time"

# This else statement just sends an echo saying that there is an error code

            else
                  echo -e "\n[WARNING]: Received HTTP status code
${server_response}"
            fi


      done <<< ${Ensembl_file}

      echo -e "\n============================================================="
      echo -e "Uniprot Data Extraction Complete"
      echo -e "=============================================================\n"

# Here is the start of the NCBI information extraction process

      echo -e "\n============================================================="
      echo -e "Starting Data Extraction from the NCBI Database"
      echo -e "=============================================================\n"

# Creation of NCBI output text file

      touch NCBI.txt

# This while loop reads the Ensembl text file and assigns a variable to each

# field

      while read biotype display_name end id seq_region_name species start strand
      do
            echo -e "\n[INFO] Processing GENE ID: ${display_name} and ORGANISM:
${species}"
            echo -e "\nQuerying NCBI API for ${display_name} and ${species}
data..."

# This code is utilising the E-utilities Api from NCBI to search the Gene

# database based on a specific query, subsequently the data are fetched into an

# xml format and then using xtract the fields that you want to be specified are

# extracted into NCBI.txt in a tabular form.

            esearch -db Gene -query "${display_name}[GENE] AND
${species}[ORGANISM]" < /dev/null | efetch -format xml | xtract -pattern
Entrezgene -element Gene-ref_locus -block Entrezgene_summary -element
Entrezgene_summary >> NCBI.txt

# The use of < /dev/null is necessary because without it the xtract stops

# interacting with the while loop

            echo -e "\n[SUCCESS] Data extracted for ${display_name} and
${species}. Saving to NCBI.txt."
            echo -e "\n-------------------------------------------------------------
```

```
--------------------------------------"


       done <<< "${Ensembl_file}"

       echo -e "\n================================================================="
       echo -e "NCBI Data Extraction Complete"
       echo -e "=================================================================\n"

# Here is the start of the MySQL Database

       echo -e "\n================================================================="
       echo -e "Starting Creation of MySQL Database"
       echo -e "=================================================================\n"

       echo -e "\n[INFO] Preparing to create MySQL database and tables..."
       echo -e "\n[INFO] Please ensure your MySQL credentials are correct.\n"

# Here we ask the user for what username and password to use

# The "-s" silences the user input thus, protecting the integrity of the database

       read -p $'What is your MySQL username (Hint: s2748062): ' mysql_username

       read -s -p $'\nWhat is your password (Hint: awxcKPiI): ' mysql_password


      read -p $'\nWhat would you like to name your database: ' n_database
# Here the code makes sure the database is clear and that it exists

       echo "DROP DATABASE ${n_database}; CREATE DATABASE ${n_database};" | mysql
-u${mysql_username} -p${mysql_password}

# Here, the creation of the ENSEMBL table begins. The columns are named, and the

# data types are specified. Furthermore, the file containing the information is

# loaded into the tables created, with the field separators specified.

       echo -e "\n-------------------------------------------------------------------
---------------------------------\n"
       echo -e "[INFO] Creating table: ENSEMBL"
       echo -e "\n[INFO] Loading data into ENSEMBL table from Ensembl.txt.\n"

       echo "DROP TABLE IF EXISTS ENSEMBL; CREATE TABLE ENSEMBL (biotype
VARCHAR(20), display_name VARCHAR(50), end INT, id VARCHAR(50) PRIMARY KEY,
seq_region_name VARCHAR(10), species VARCHAR(50), start INT, strand TINYINT);
LOAD DATA LOCAL INFILE './Ensembl.txt' INTO TABLE ENSEMBL FIELDS TERMINATED BY
'\t' LINES TERMINATED BY '\n';" | mysql ${n_database} -u${mysql_username} -
p${mysql_password}


# Here, the creation of the NCBI table begins. The columns are named, and the

# data types are specified. Furthermore, the file containing the information is

# loaded into the tables created, with the field separators specified.

       echo -e "\n-------------------------------------------------------------------
---------------------------------\n"
```

```
      echo -e "[INFO] Creating table: NCBI"
      echo -e "\n[INFO] Loading data into NCBI table from NCBI.txt.\n"

      echo "DROP TABLE IF EXISTS NCBI; CREATE TABLE NCBI (symbol VARCHAR(50),
Gene_description TEXT, PRIMARY KEY (symbol)); LOAD DATA LOCAL INFILE './NCBI.txt'
INTO TABLE NCBI FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n';" | mysql
${n_database} -u${mysql_username} -p${mysql_password}

# Here, the creation of the Uniprot table begins. The columns are named, and the

# data types are specified. Furthermore, the file containing the information is

# loaded into the tables created, with the field separators specified.

      echo -e "\n-----------------------------------------------------------------
---------------------------------\n"
      echo -e "[INFO] Creating table: Uniprot"
      echo -e "\n[INFO] Loading data into Uniprot table from Uniprot.txt.\n"

      echo "DROP TABLE IF EXISTS UNIPROT; CREATE TABLE UNIPROT (accession
VARCHAR(20), gene_primary VARCHAR(100), organism_name VARCHAR(100), cc_function
TEXT, Protein_Name TEXT, sequence TEXT, PRIMARY KEY (accession)); LOAD DATA LOCAL
INFILE './Uniprot.txt' INTO TABLE UNIPROT FIELDS TERMINATED BY '\t' LINES
TERMINATED BY '\n';" | mysql ${n_database} -u${mysql_username}  -
p${mysql_password}

      echo -e "\n-----------------------------------------------------------------
---------------------------------\n"
      echo -e "[SUCCESS] All database tables have been successfully created and
populated."

# Here, we display the combined view of all the data extracted, with a slightly

# shortened version in a vertical view.

      echo -e "\n-----------------------------------------------------------------
---------------------------------\n"
      echo -e "[INFO] Generating final combined view of extracted data..."
      echo -e "\n[INFO] Displaying a shortened version below for easier
reading.\n"

      echo -e "SELECT ENSEMBL.*, SUBSTRING(NCBI.Gene_description, 1, 600) AS
Gene_description, SUBSTRING(UNIPROT.accession, 1, 10) AS accession,
SUBSTRING(UNIPROT.cc_function, 1, 1000) AS cc_function,
SUBSTRING(UNIPROT.Protein_Name, 1, 200) AS Protein_Name,
SUBSTRING(UNIPROT.sequence, 1, 100) AS Short_Sequence FROM ENSEMBL LEFT JOIN NCBI
ON TRIM(ENSEMBL.display_name) = TRIM(NCBI.symbol) LEFT JOIN UNIPROT ON
TRIM(ENSEMBL.display_name) = TRIM(UNIPROT.gene_primary) \G"| mysql ${n_database}
-u${mysql_username} -p${mysql_password}

# Outputting the table in a tsv so when imported in excel it looks better than a

# csv

       echo -e "\n-----------------------------------------------------------------
---------------------------------\n"
      echo -e "SELECT ENSEMBL.*, NCBI.Gene_description, UNIPROT.accession,
UNIPROT.cc_function, UNIPROT.Protein_Name, UNIPROT.sequence FROM ENSEMBL LEFT
JOIN NCBI ON TRIM(ENSEMBL.display_name) = TRIM(NCBI.symbol) LEFT JOIN UNIPROT ON
```

```
TRIM(ENSEMBL.display_name) = TRIM(UNIPROT.gene_primary)" | mysql ${n_database}
-u${mysql_username} -p${mysql_password} > FINAL.tsv

      echo -e "\n[INFO] Final data output created: FINAL.tsv"
      echo -e "\nYou can view the full dataset in this file."

      echo -e "\n============================================================="
      echo -e "Thank you for using DatabaseMaker2025!\n"
      echo -e "Script execution complete. Have a great day!"
      echo -e "=============================================================\n"

else
      echo -e "\nExiting Script...\n"
fi
```

Supplementary Data 2: MySQL Custom Database Schema

```
CREATE DATABASE IF NOT EXISTS `databases_project`
  DEFAULT CHARACTER SET utf8mb4
  DEFAULT COLLATE utf8mb4_0900_ai_ci;

USE `databases_project`;

-- ENSEMBL
DROP TABLE IF EXISTS `ENSEMBL`;
CREATE TABLE `ENSEMBL` (
  `biotype`        varchar(20)  DEFAULT NULL,
  `display_name`   varchar(50)  DEFAULT NULL,
  `end`            int          DEFAULT NULL,
  `id`             varchar(50)  NOT NULL,
  `seq_region_name` varchar(10) DEFAULT NULL,
  `species`        varchar(50)  DEFAULT NULL,
  `start`          int          DEFAULT NULL,
  `strand`         tinyint      DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- NCBI
DROP TABLE IF EXISTS `NCBI`;
```

```
CREATE TABLE `NCBI` (
  `symbol`           varchar(50) NOT NULL,
  `Gene_description` text,
  PRIMARY KEY (`symbol`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


-- UNIPROT
DROP TABLE IF EXISTS `UNIPROT`;
CREATE TABLE `UNIPROT` (
  `accession`     varchar(20)  NOT NULL,
  `gene_primary`  varchar(100) DEFAULT NULL,
  `organism_name` varchar(100) DEFAULT NULL,
  `cc_function`   text,
  `Protein_Name`  text,
  `sequence`      text,
  PRIMARY KEY (`accession`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

6. References

Bash loop using Edirect (esearch|efetch). (n.d.). Retrieved November 11, 2024, from https://www.biostars.org/p/217450/

Ensembl Rest API - GET lookup/id/:id. (n.d.). Retrieved November 11, 2024, from https://rest.ensembl.org/document

Ensembl Stable ID Format. (n.d.). Retrieved November 11, 2024, from https://www.ensembl.org/info/genome/stable_ids/index.html#:~:text=Stable%20IDs%20are%20created%20in,what%20species%20they%20are%20in

Jonathan Kans, PhD. (2013, April 13). *Entrez Direct: E-utilities on the Unix Command Line*. https://www.ncbi.nlm.nih.gov/books/NBK179288/

Programmatic access - Retrieving entries via queries. (n.d.). Retrieved November 11, 2024, from https://www.uniprot.org/help/api_queries

UniProtKB Return Fields. (n.d.). Retrieved November 11, 2024, from https://www.uniprot.org/help/return_fields