PROJECT PHASE II- DATABASES (in my office, by Googly appointment)

What are you going to do during your presentation?

(1)Each project member will have a **separate** time window of 20 minutes to present your system to me in English. I will whatsapp you a Doodle to schedule your appointment.

(2) You can bring along your own cheat-sheet if you prefer.

(3)You will be asked to run several SQL commands on your database with flexible inputs that I will provide. Your database outputs should be obtained from **at least 1 stored procedure (with input/output variables IN/OUT/INOUT) and 1 view.**
(4)Application interface is bonus if otherwise stated as required in your feedback. Please provide your bitbucket/github address of your GUI and screenshots in your report

Requirements for your project report:

(1) On your cover page include the names of your team members and a 2-3 sentence description of your Project

Team members
Isa Öselmiş, 200709004
Batuhan Kırma, 200709070
Çilem Özdemir, 234242001

Dataset is name “Genetic Variant Classifications”. Our data set includes genetic variations occurring in cells (single nucleotide variants, insertions, deletions, etc.) and their relationships with phenotype/disease. Our aim organizing the dataset containing genetic variants, making it possible to query variants according to various criteria, present new variants to researchers by determining the disease relationship of the variants.

(2) Describe any changes/additions if your project description or database design has changed since your Phase I submission.

We ensured that our table was in the third normal form (3NF) by fixing the errors in our ER diagram. We have optimally combined the data in our dataset and loaded it into our ER diagram.

(3) Briefly describe how you loaded the database with values including the sources of your data (e g URL s) and pointers to any specialized code you developed for pre- processing, extracting or loading the data. Did you upload all your data? If not, why, state it here.

https://www.kaggle.com/datasets/kevinarvai/clinvar-conflicting

Python pandas library was used to organize the data. The data was loaded as CSV files using the following sql code for each CSV.

```
LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘filename.csv'
INTO TABLE variant_has_variant_feature
CHARACTER SET utf8
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

(4) If you did not use MySql Workbench on Windows, describe the exact software/hardware platform you used.

We used MySql Workbench on Windows.

(5) Provide a brief users guide describing in sufficient detail what each View/Stored procedure does in English.

VIEW1 (cancervariants): Shows the identification numbers of variants associated with cancer disease in the snpdb database. This view can be used to visualize cancer-associated variants without specifying any conditions.

VIEW2 (tp53_variants_prediction): Provides a view in which the possible effects of variants related to the TP53 gene in the snpdb database are predicted on the protein. With this view, it can be used to predict the impact of possible variants in the TP53 gene, an important gene that plays key roles in cellular processes.

IN STORED: The stored procedure GetVariantIdByGeneSymbol uses gene symbols as input parameters. The aim is to retrieve information about variants related to any gene in the database based on the gene symbol.

OUT STORED: The stored procedure GetVariantIdByGeneSymbol uses an input parameter, Symbol, of type varchar (255) and an output parameter, variantCount. The aim is to determine the total number of variants of the specified gene. The specified total number is stored in the output parameter.

INOUT STORED: The stored procedure GetDiseaseNameFromIdentifier uses an INOUT parameter of type varchar (255). Searches the diseaseNames directory using the value of the identifier parameter.

(6) Include the outputs of your favorite/interesting/challenging stored procedures(with IN, OUT and INOUT)/views.


view.csv


store_procedure.csv

(7) Give an analysis of your system's limitations and list suggested possibilities for improvement.

Some VARCHAR fields have arbitrary lengths (e.g., VARCHAR(145)), which might lead to inefficient storage.Use appropriate data types and lengths based on the actual expected data. Optimize storage where possible.

(8) Submit a full relational table specification of your database in the SQL Database Definition Language (DDL) This specification should include both the data type of each attribute the not null constraint when appropriate and sample data values for each attribute represented as comments You should also specify the primary keys (e g primary key (ssn)) and referential constraints (e g foreign key (mgrssn) references employee(ssn)) Many groups included this specification in their project proposal.

variant_type
variant_type (variant_type_id INT NOT NULL AUTO_INCREMENT, variant_type_name VARCHAR(45) NOT NULL, PRIMARY KEY (variant_type_id));

variant
variant (variant_id INT NOT NULL AUTO_INCREMENT, chromosome_position INT NULL, reference_allele VARCHAR(145) NULL, alternate_allele VARCHAR(145) NULL, cdna_start_position INT NULL, cdna_stop_position INT NULL, cds_start_position INT NULL, cds_stop_position INT NULL, protein_start_position INT NULL, protein_stop_position INT NULL, reference_aa CHAR(1) NULL, alternate_aa CHAR(1) NULL, reference_codon VARCHAR(145) NULL, alternate_codon VARCHAR(145) NULL, variant_type_id INT NOT NULL, exon_number

INT NULL, intron_number INT NULL, PRIMARY KEY (variant_id), CONSTRAINT fk_variant_variant_type1 FOREIGN KEY (variant_type_id) REFERENCES variant_type (variant_type_id));

allele_frequency_db
allele_frequency_db (db_id INT NOT NULL, db_name VARCHAR(45) NULL, PRIMARY KEY (db_id));

variant_has_allele_frequency
variant_has_allele_frequency (variant_id INT NOT NULL, allele_frequency_db_id INT NOT NULL, frequency FLOAT NULL, PRIMARY KEY (variant_id, allele_frequency_db_id), CONSTRAINT fk_variant_has_allele_frequency_db_variant FOREIGN KEY (variant_id) REFERENCES variant (variant_id) CONSTRAINT fk_variant_has_allele_frequency_db_allele_frequency_db1 FOREIGN KEY (allele_frequency_db_id) REFERENCES allele_frequency_db (db_id));

variant_feature
variant_feature (variant_feature_id INT NOT NULL, variant_feature_name VARCHAR(45) NULL, PRIMARY KEY (variant_feature_id));

variant_has_variant_feature
variant_has_variant_feature (variant_id INT NOT NULL, variant_feature_id INT NOT NULL, PRIMARY KEY (variant_id, variant_feature_id), CONSTRAINT fk_variant_has_variant_feature_variant1 FOREIGN KEY (variant_id) REFERENCES variant (variant_id), CONSTRAINT fk_variant_has_variant_feature_variant_feature1 FOREIGN KEY (variant_feature_id) REFERENCES variant_feature (variant_feature_id));

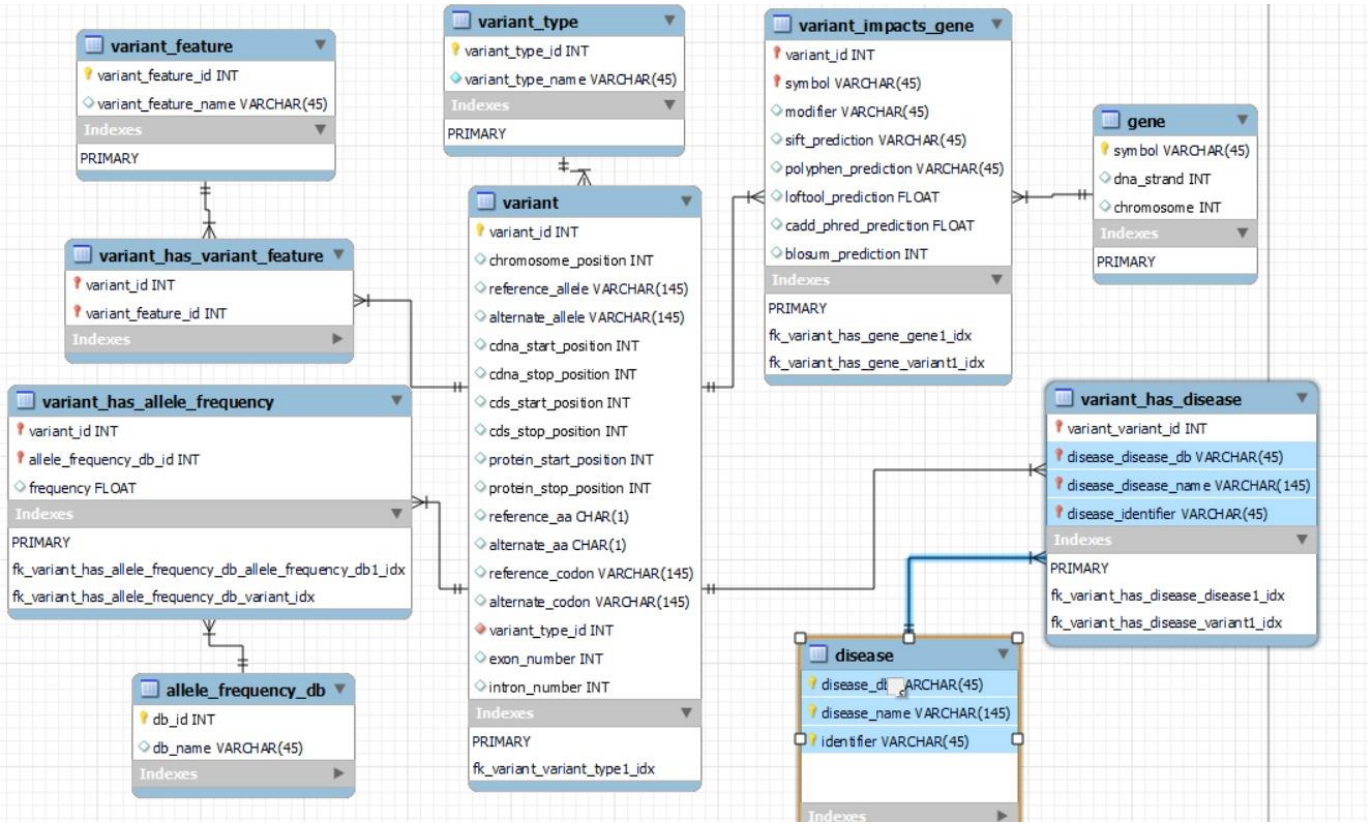
gene
gene (symbol VARCHAR(45) NOT NULL, dna_strand INT NULL, chromosome INT NULL, PRIMARY KEY (symbol));

variant_impacts_gene
variant_impacts_gene (variant_id INT NOT NULL, symbol VARCHAR(45) NOT NULL, modifier VARCHAR(45) NULL, sift_prediction VARCHAR(45) NULL, polyphen_prediction VARCHAR(45) NULL, loftool_prediction FLOAT NULL, cadd_phred_prediction FLOAT NULL, blosum_prediction INT NULL, PRIMARY KEY (variant_id, symbol), CONSTRAINT fk_variant_has_gene_variant1 FOREIGN KEY (variant_id) REFERENCES variant (variant_id), CONSTRAINT fk_variant_has_gene_gene1 FOREIGN KEY (symbol) REFERENCES gene (symbol));

disease

disease (disease_db VARCHAR(45) NOT NULL, disease_name VARCHAR(145) NOT NULL, identifier VARCHAR(45) NOT NULL, PRIMARY KEY (disease_db, disease_name, identifier));

variant_has_disease
variant_has_disease (variant_variant_id INT NOT NULL, disease_disease_db VARCHAR(45) NOT NULL, disease_disease_name VARCHAR(145) NOT NULL, disease_identifier VARCHAR(45) NOT NULL, PRIMARY KEY (variant_variant_id, disease_disease_db, disease_disease_name, disease_identifier), CONSTRAINT fk_variant_has_disease_variant1 FOREIGN KEY (variant_variant_id) REFERENCES variant (variant_id), CONSTRAINT fk_variant_has_disease_disease1 FOREIGN KEY (disease_disease_db, disease_disease_name , disease_identifier) REFERENCES disease (disease_db , disease_name , identifier));



(9) Include all your SQL code used in your system as well as any additional Php/Perl/Python/Java/SQL-Loader programs you used for data acquisition and input to your presentation. If you have long sequences of input statements in excess of 2 pages provide a 2-3 page representative sample.

VIEW

```
CREATE VIEW CancerVariants AS
SELECT variant.variant_id, variant_has_disease.disease_disease_name AS disease_name
FROM variant
JOIN variant_has_disease ON variant.variant_id = variant_has_disease.variant_variant_id
WHERE variant_has_disease.disease_disease_name LIKE ("%CANCER%");
```

STORED PROCEDURE-IN

```
CREATE PROCEDURE GetVariantIdByGeneSymbol(IN geneSymbol VARCHAR(255))
BEGIN
   SELECT variant_impacts_gene.variant_id
   FROM variant_impacts_gene
   JOIN gene ON variant_impacts_gene.symbol = gene.symbol
   WHERE gene.symbol = geneSymbol;
END //
DELIMITER ;

call GetVariantIdByGeneSymbol ("TP53");
```

STORED PROCEDURE-OUT

```
DELIMITER //
CREATE PROCEDURE GetVariantCountByGeneSymbol(IN geneSymbol VARCHAR(255), OUT variantCount INT)
BEGIN
```

```
    SELECT count(variant_impacts_gene.variant_id)
    INTO variantCount
            FROM variant_impacts_gene
            JOIN gene ON variant_impacts_gene.symbol = gene.symbol
            WHERE gene.symbol = geneSymbol;
END //
DELIMITER ;

select @totalCountOCA;
call GetVariantCountByGeneSymbol("oca", @totalCountOCA);
```

STORED PROCEDURE-INOUT

```
DELIMITER //

CREATE PROCEDURE GetDiseaseNameFromIdentifier(INOUT diseaseInput VARCHAR(255))
BEGIN
    select disease.disease_name
    into diseaseInput
            from disease
            where disease.identifier = diseaseInput;
END //

DELIMITER ;
set @diseaseInputOutput = "EFO_0004269\r";
call GetDiseaseNameFromIdentifier(@diseaseInputOutput);
select @diseaseInputOutput;
```

LOADING DATA

#Create CSV files/Split Codes

```
import csv
from typing import Any

PATH_DATASET_CSV = "./variant_has_feature.csv"
PATH_VARIANT_HAS_FEATURE_CSV = "./variant_has_featuree.csv"

def read_csv(file_path: str) -> list[list[Any]]:
    with open(file_path, "r", encoding="latin1") as file:
        reader = csv.reader(file, delimiter=";")
        return [row for row in reader][1:][:100]

def create_csv(file_path: str, rows: list[list[Any]]):
    with open(file_path, "w", newline="") as file:
        writer = csv.writer(file, delimiter=",")
        writer.writerows(rows)

variant_has_feature_rows = []
rows = read_csv(PATH_DATASET_CSV)
for row in rows:
    for mc in row[1].split(","):
        variant_has_feature_rows.append([row[0], mc])

create_csv(PATH_VARIANT_HAS_FEATURE_CSV, variant_has_feature_rows)

import csv
from typing import Any
```

```
PATH_DATASET_CSV = "./dataset.csv"
PATH_DISEASE_CSV = "./disease.csv"
COL_DISEASE = 9
COL_DISEASE_DB = 7

def read_csv(file_path: str) -> list[list[Any]]:
    with open(file_path, "r", encoding="latin1") as file:
        reader = csv.reader(file, delimiter=";")
        return [row for row in reader][1:][:100]

def create_csv(file_path: str, rows: list[list[Any]]):
    with open(file_path, "w", newline="") as file:
        writer = csv.writer(file, delimiter=",")
        writer.writerows(rows)

disease_rows = []
rows = read_csv(PATH_DATASET_CSV)
for row in rows:
    diseases_splitted = row[COL_DISEASE].split("|")
    disease_dbs_splitted = row[COL_DISEASE_DB].split("|")
    for disease_index, disease_dbs in enumerate(disease_dbs_splitted):
        for disease_db_disease_id in disease_dbs.split(","):
            disease_db_disease_id = disease_db_disease_id.split(":")
            disease_row = [disease_db_disease_id[0], diseases_splitted[disease_index], disease_db_disease_id[-1]]
            if disease_row not in disease_rows:
                disease_rows.append(disease_row)

create_csv(PATH_DISEASE_CSV, disease_rows)
import csv
from typing import Any

PATH_DISEASE_CSV = "./disease.csv"
PATH_NEW_DISEASE_CSV = "./new_disease.csv"

def read_csv(file_path: str) -> list[list[Any]]:
    with open(file_path, "r", encoding="latin1") as file:
        reader = csv.reader(file, delimiter=";")
        return [row for row in reader][1:]

def create_csv(file_path: str, rows: list[list[Any]]):
    with open(file_path, "w", newline="") as file:
        writer = csv.writer(file, delimiter=",")
        writer.writerows(rows)

disease_rows = []
rows = read_csv(PATH_DISEASE_CSV)
for row in rows:
    row_splitted = row[0].split(",")
    try:
        disease_splitted = row[0].split("\"")[1]
    except IndexError as ex:
        disease_rows.append(row_splitted)
        continue
    disease = disease_splitted.replace(",", "")
    disease_rows.append([row_splitted[0], disease, row_splitted[-1]])

create_csv(PATH_NEW_DISEASE_CSV, disease_rows)
PATH_VARIANT_HAS_DISEASE_CSV = "./variant_has_disease.csv"
PATH_NEW_VARIANT_HAS_DISEASE_CSV = "./new_variant_has_disease.csv"
```

```
def read_csv(file_path: str) -> list[list[Any]]:
    with open(file_path, "r", encoding="latin1") as file:
        reader = csv.reader(file, delimiter=";")
        return [row for row in reader][1:]

def create_csv(file_path: str, rows: list[list[Any]]):
    with open(file_path, "w", newline="") as file:
        writer = csv.writer(file, delimiter=",")
        writer.writerows(rows)

variant_has_disease = []
rows = read_csv(PATH_VARIANT_HAS_DISEASE_CSV)
for row in rows:
    row_splitted = row[0].split(",")
    try:
        disease_splitted = row[0].split("\"")[2]
    except IndexError as ex:
        variant_has_disease.append(row_splitted)
        continue
    disease = disease_splitted.replace(",", "")
    variant_has_disease.append([row_splitted[0], disease, row_splitted[-1]])

create_csv(PATH_NEW_VARIANT_HAS_DISEASE_CSV, variant_has_disease)

PATH_DATASET_CSV = "./dataset.csv"
PATH_VARIANT_HAS_DISEASE_CSV = "./variant_has_disease.csv"
COL_DISEASE_DB = 7
COL_DISEASE = 9

def read_csv(file_path: str) -> list[list[Any]]:
    with open(file_path, "r", encoding="latin1") as file:
        reader = csv.reader(file, delimiter=";")
        return [row for row in reader][1:]

def create_csv(file_path: str, rows: list[list[Any]]):
    with open(file_path, "w", newline="") as file:
        writer = csv.writer(file, delimiter=",")
        writer.writerows(rows)

variant_has_disease_rows = []
rows = read_csv(PATH_DATASET_CSV)
for variant_id, row in enumerate(rows):
    variant_id += 1
    diseases_splitted = row[COL_DISEASE].split("|")
    disease_dbs_splitted = row[COL_DISEASE_DB].split("|")
    for disease_index, disease_dbs in enumerate(disease_dbs_splitted):
        for disease_db_disease_id in disease_dbs.split(","):
            disease_db_disease_id = disease_db_disease_id.split(":")

            disease_db = disease_db_disease_id[0]
            disease = diseases_splitted[disease_index]
            disease_identifier = disease_db_disease_id[-1]
            variant_has_disease_rows.append([variant_id, disease_db, disease, disease_identifier])

create_csv(PATH_VARIANT_HAS_DISEASE_CSV, variant_has_disease_rows)
```

#Loading data to MySql

```
load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘variant.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘allele_frequency_db.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘disease.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘gene.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘variant_feature.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘variant_has_allele_frequency.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘variant_has_disease.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';


load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘variant_has_variant_feature.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';
load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘variant_impacts_gene.csv'
into table variant_has_variant_feature
character set utf8
```

```
fields terminated by ','
lines terminated by '\n';

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\ ‘variant_type.csv'
into table variant_has_variant_feature
character set utf8
fields terminated by ','
lines terminated by '\n';
```