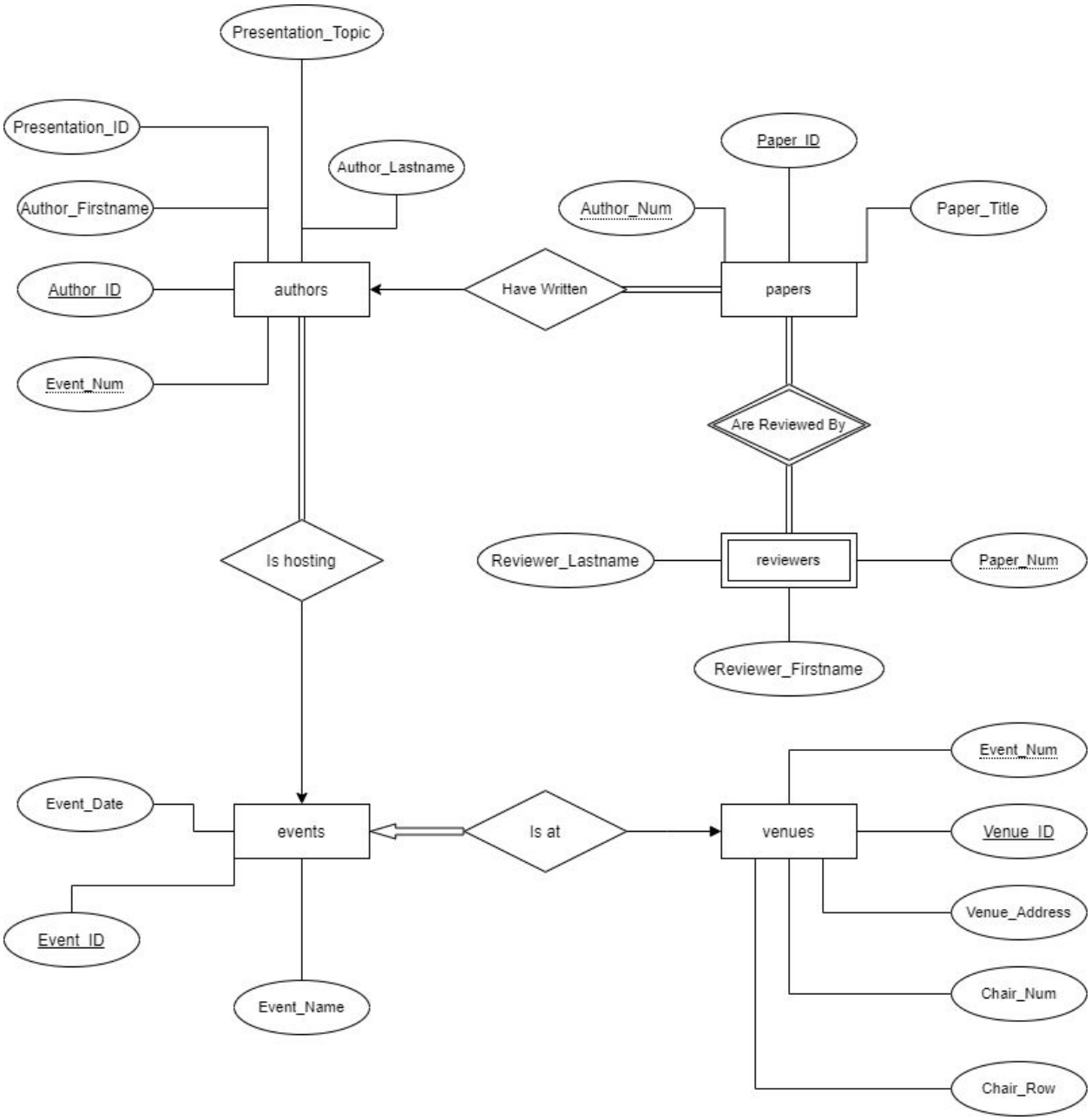
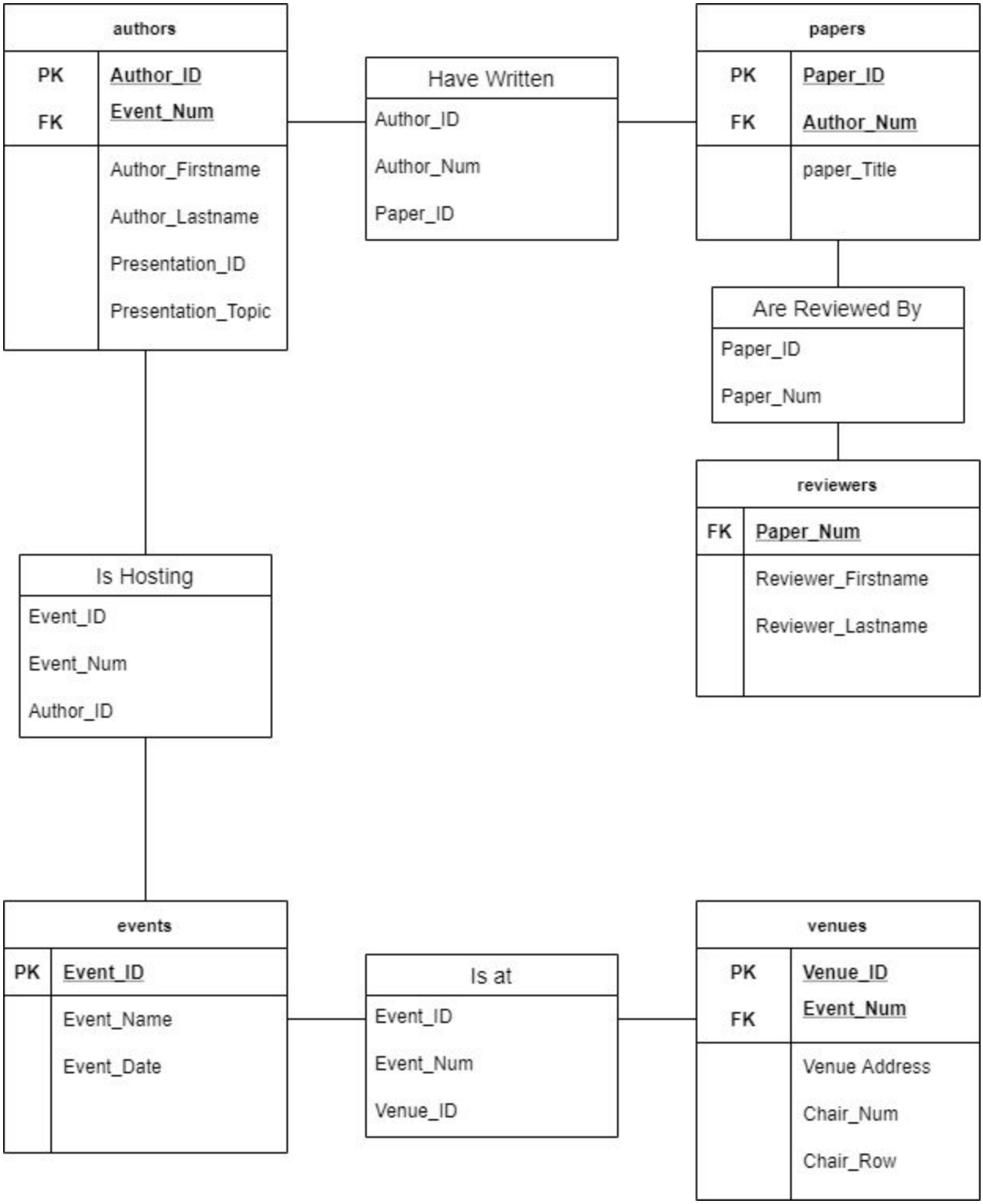
1. Draw its E-R diagram with the notation used in class.



2. Give a relational representation (with a set of relations and a relational diagram) of the E-R diagram that allows for 2 functional preserving and lossless join decompositions (i.e., first version not yet optimized and fully decomposed). Describe its functional dependencies.



**Functional Dependencies:**

events = (Event_ID, Event_Name, Event_Date)
FD = {Event_ID --> Event_Name
Event_ID --> Event_Date}

authors = (Event_Num, Author_ID, Author_FIrstname, Author_Lastname, Presentation_ID, Presentation_Topic)
FD = {Event_Num --> Author_ID
Author_ID --> Author_Firstname
Author_ID --> Author_Lastname
Author_ID --> Presentation_ID
Presentation_ID --> Presentation_Topic}

papers = (Author_Num, Paper_ID, paper_Title)
FD = {Author_Name --> Paper_ID
Paper_ID --> paper_Title}

reviewers = (Paper_Num, Reviewer_Firstname, Reviewer_Lastname)
FD = {Paper_Num --> Reviewer_Firstname,
Paper_Num --> Reviewer_Lastname}

venues = (Event_Num, Venue_ID, Venue_Address, Chair_Num, Chair_Row)
FD = {Event_Num --> Venue_ID
Venue_ID --> Venue_Address
Venue_ID --> Chair_Num
Chair_Num --> Chair_Row}

3. Give the two decompositions you select for further steps, and prove that they are lossless join and functional preserving.

## Decomposition 1:

For the first decomposition I decompose the table venues into venues and chairs.

venues = (Event_Num, Venue_ID, Venue_Address)
FD = {Event_Num --> Venue_ID
Venue_ID --> Venue_Address}

chairs = (Venue_ID, Chair_Num, Chair_Row)
FD = {Venue_ID --> Chair_Num
Chair_Num --> Chair_Row}

This is a lossless join decomposition because a natural join would give the original relation. In these examples the original venues table will be R, the new venues table will be $R_1$, and the table chairs will be $R_2$. To prove that this decomposition is lossless join it must satisfy 3 requirements.

a) The union of attributes of venues ($R_1$) and chairs($R_2$) are equal to the attributes in venues (R):

venues(Event_Num, Venue_ID, Venue_Address) U chairs(Venue_ID, Chair_Num, Chair_Row) = venues = (Event_Num, Venue_ID, Venue_Address, Chair_Num, Chair_Row)

Or in other words

$R_1$ U $R_2$ = R so this condition is fulfilled.

b) The intersection of attributes in $R_1$ and $R_2$ are not null or in other words, there must be some common attribute which is present in both decomposed relations:

venues(Event_Num, Venue_ID, Venue_Address) ∩ chairs(Venue_ID, Chair_Num, Chair_Row) = Venue_ID

Or in other words if Venue_ID = A

$R_1$ ∩ $R_2$ = A so this condition is fulfilled.

c) The common attribute must be a key for at least one relation in $R_1$ or $R_2$.

The common attribute Venue_ID is the primary key in $R_1$ and the foreign key in $R_2$ so this condition is fulfilled.

This is also a functional dependency preserving decomposition because all the dependencies in venues (R) are in venues ($R_1$) and chairs ($R_2$).

For ease the functional dependencies will be renamed as:

Event_Num --> Venue_ID will be changed to (A -> B)
Venue_ID --> Venue_Address will be changed to (B -> C)
Venue_ID --> Chair_Num will be changed to (B -> D)
Chair_Num --> Chair_Row will be changed to (D -> E)

In Venues ($R_1$) the functional dependencies are:
(A -> B)
(B -> C)

and in venues ($R_2$) the functional dependencies are:
(B -> D)
(D -> E)

All of the functional dependencies of R are found in $R_1$ or R , therefore this decomposition is functional preserving.

## Decomposition 2:

For the second decomposition I decompose the table authors into authors and presentations. Furthermore I will remove redundancy by getting rid of the attributes Author_Firstname and Author_Lastname and adding the attribute Author_Name.

authors = (Event_Num, Author_ID, Author_Name)
FD = {Event_Num --> Author_ID
Author_ID --> Author_Name}

presentations = (Author_ID, Presentation_ID, Presentation_Topic)
FD = {Author_ID --> Presentation_ID
Presentation_ID --> Presentation_Topic}

This is a lossless join decomposition because a natural join would give the original relation. In these examples the original authors table will be R, the new authors table will be $R_1$, and the table presentations will be $R_2$. To prove that this decomposition is lossless join it must satisfy 3 requirements.

a) The union of attributes of authors($R_1$) and presentations($R_2$) are equal to the attributes in authors(R):

authors = (Event_Num, Author_ID, Author_Name) U presentations = (Author_ID, Presentation_ID, Presentation_Topic) = authors = (Event_Num, Author_ID, Author_Name, Presentation_ID, Presentation_Topic)

Or in other words

$R_1$ U $R_2$ = R so this condition is fulfilled.

b) The intersection of attributes in $R_1$ and $R_2$ are not null or in other words, there must be some common attribute which is present in both decomposed relations:

authors = (Event_Num, Author_ID, Author_Name) ∩ presentations = (Author_ID, Presentation_ID, Presentation_Topic) = Author_ID

Or in other words if Author_ID = A

$R_1$ ∩ $R_2$ = A so this condition is fulfilled.

c) The common attribute must be a key for at least one relation in $R_1$ or $R_2$.

The common attribute Author_ID is the primary key in $R_1$ and the foreign key in $R_2$ so this condition is fulfilled. This is also a functional dependency preserving decomposition because all the dependencies in venues (R) are in venues ($R_1$) and chairs ($R_2$).

For ease the functional dependencies will be renamed as:

| | |
|---|---|
| Event_Num --> Author_ID | will be changed to (A -> B) |
| Author_ID --> Author_Name | will be changed to (B -> C) |
| Author_ID --> Presentation_ID | will be changed to (B -> D) |
| Presentation_ID --> Presentation_Topic | will be changed to (D -> E) |

In Venues ($R_1$) the functional dependencies are:
(A -> B)
(B -> C)

and in venues ($R_2$) the functional dependencies are:
(B -> D)
(D -> E)

All of the functional dependencies of R are found in $R_1$ or R2, therefore this decomposition is functional preserving.

4. d.ddl, d2.ddl, and d3.ddl are all included in the .zip archive.

5. The first version of the database was filled with the scripts provided in the .zip archive.

6. Give the SQL queries to copy the data from the first version into its 2 decompositions.

```
>sqlite3 shoop_meyer_project_1.sqlite
>ATTACH shoop_meyer_project_2.sqlite as new
>CREATE TABLE new.events(
    Event_ID INTEGER PRIMARY KEY,
    Event_Name TEXT,
    Event_Date DATE
);
>INSERT INTO new.events SELECT * FROM events;
>CREATE TABLE new.venues(
    Venue_ID INTEGER PRIMARY KEY,
    Venue_Address TEXT,
    Event_Num INTEGER,
    FOREIGN KEY (Event_Num)
        REFERENCES events (Event_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.venues SELECT Venue_ID, Venue_Address, Event_Num FROM events;
>CREATE TABLE new.chairs(
    Chair_Num INTEGER PRIMARY KEY,
    Chair_Row INTEGER,
    Venue_ID INTEGER,
    FOREIGN KEY (Venue_ID)
    REFERENCES venues (Venue_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
>INSERT INTO new.chairs SELECT Chair_Num, Chair_Row, Venue_ID FROM venues;
>CREATE TABLE new.authors(
    Author_ID INTEGER PRIMARY KEY,
    Author_Firstname TEXT,
    Author_Lastname TEXT,
    Presentation_ID INTEGER,
    Presentation_Topic TEXT,
    Event_Num INTEGER,
    FOREIGN KEY (Event_Num)
        REFERENCES events (Event_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.authors SELECT * FROM authors;
>CREATE TABLE new.papers(
    Paper_ID INTEGER PRIMARY KEY,
    Paper_Title TEXT,
    Author_Num INTEGER,
    FOREIGN KEY (Author_Num)
        REFERENCES authors (Author_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.papers SELECT * FROM papers;
>CREATE TABLE new.reviewers(
    Reviewer_Firstname TEXT,
    Reviewer_Lastname TEXT,
    Paper_Num INTEGER,
    FOREIGN KEY (Paper_Num)
        REFERENCES papers (Paper_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.reviewers SELECT * FROM reviewers;



>sqlite3 shoop_meyer_project_2.sqlite
>ATTACH shoop_meyer_project_3.sqlite AS new
>CREATE TABLE events(
    Event_ID INTEGER PRIMARY KEY,
    Event_Name TEXT,
    Event_Date DATE
);
>INSERT INTO new.events SELECT * FROM events;
>CREATE TABLE venues(
    Venue_ID INTEGER PRIMARY KEY,
    Venue_Address TEXT,
    Event_Num INTEGER,
    FOREIGN KEY (Event_Num)
        REFERENCES events (Event_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.venues SELECT * FROM venues;
>CREATE TABLE chairs(
    Chair_Num INTEGER PRIMARY KEY,
    Chair_Row INTEGER,
```

```
    Venue_ID INTEGER,
    FOREIGN KEY (Venue_ID)
    REFERENCES venues (Venue_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
>INSERT INTO new.chairs SELECT * FROM chairs;
>CREATE TABLE authors(
    Author_ID INTEGER PRIMARY KEY,
    Author_Name TEXT,
    Event_Num INTEGER,
    FOREIGN KEY (Event_Num)
        REFERENCES events (Event_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.authors SELECT Author_ID, Author_Name, Event_Num FROM authors;
>CREATE TABLE presentations(
    Presentation_ID INTEGER,
    Presentation_Topic TEXT,
    Author_ID INTEGER,
    FOREIGN KEY (Author_ID)
    REFERENCES authors (Author_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
>INSERT INTO new.presentations SELECT Presentation_ID, Presentation_Topic, Author_ID FROM
authors;
>CREATE TABLE papers(
    Paper_ID INTEGER PRIMARY KEY,
    Paper_Title TEXT,
    Author_Num INTEGER,
    FOREIGN KEY (Author_Num)
        REFERENCES authors (Author_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.papers SELECT * FROM papers;
>CREATE TABLE reviewers(
    Reviewer_Firstname TEXT,
    Reviewer_Lastname TEXT,
    Paper_Num INTEGER,
    FOREIGN KEY (Paper_Num)
        REFERENCES papers (Paper_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
>INSERT INTO new.reviewers SELECT * FROM reviewers;
```

7. Propose in English 3 queries that require at least 2 table joins each and such that all tables are involved in at least 2 queries.

Query 1: Select the attributes for the event name, venue address and chair number. This will utilize a table join from events and venues in the original database. It will also require a further table join between venues and chairs in the first and second decomposition.

Query 2: Select the attributes for the authors name and from each author select each presentation topic, paper title, and reviewers last name. This will utilize a table join from authors and papers, and a table join from papers and reviewers in the original database and the first decomposition. In the second decomposition it will require another table join between authors and presentations.

Query 3: Select the attributes for the event name and that event's author_ID's and that author's presentation topic and paper titles where the event ID is less than 100. This will utilize a table join from authors and papers in the original database and the first decomposition. In the second decomposition this will also require a join between authors and presentations.

8. Propose SQL implementations of the 3 queries on all three versions of the database.

**QUERY 1:**

**Database 1:**

```
SELECT
        Chair_Num,
        Venue_Address,
        Event_Name
FROM
        venues
INNER JOIN events
        ON events.Event_ID = venues.Event_Num;
```

**Database 2:**

```
SELECT
        Chair_Num,
        Venue_Address,
        Event_Name
FROM
        chairs
INNER JOIN events
        ON events.Event_ID = venues.Event_Num
INNER JOIN venues
        ON venues.Venue_ID = chairs.Venue_ID;
```

**Database 3:**

```
SELECT
        Chair_Num,
        Venue_Address,
        Event_Name
FROM
        chairs
INNER JOIN events
        ON events.Event_ID = venues.Event_Num
INNER JOIN venues
        ON venues.Venue_ID = chairs.Venue_ID;
```

**QUERY 2:**

**Database 1:**

```
SELECT
        Reviewer_Lastname,
        Paper_Title,
        Presentation_Topic,
        Author_Firstname,
        Author_Lastname
FROM
        reviewers
INNER JOIN authors
        ON authors.Author_ID = papers.Author_Num
INNER JOIN papers
        ON papers.Paper_ID = reviewers.Paper_Num;
```

**Database 2:**

```
SELECT
        Reviewer_Lastname,
        Paper_Title,
        Presentation_Topic,
        Author_Firstname,
        Author_Lastname
FROM
        reviewers
INNER JOIN authors
        ON authors.Author_ID = papers.Author_Num
INNER JOIN papers
        ON papers.Paper_ID = reviewers.Paper_Num;
```

**Database 3:**

```
SELECT
        Reviewer_Lastname,
        Paper_Title,
        a1.Author_Name,
        Presentation_Topic,
        a2.Author_Name
FROM
        reviewers,
```

```
        presentations
INNER JOIN authors a1
        ON papers.Author_Num = a1.Author_ID
INNER JOIN papers
        ON papers.Paper_ID = reviewers.Paper_Num
INNER JOIN authors a2
        ON presentations.Author_ID = a2.Author_ID;
```

**QUERY 3:**

**Database 1:**

```
SELECT
        Event_Name,
        Author_ID,
        Paper_Title,
        Presentation_Topic
FROM
        papers
INNER JOIN events
        ON authors.Event_Num = events.Event_ID
INNER JOIN authors
        ON papers.Author_Num = authors.Author_ID
WHERE events.Event_ID < 100;
```

**Database 2:**

```
SELECT
        Event_Name,
        Author_ID,
        Paper_Title,
        Presentation_Topic
FROM
        papers
INNER JOIN events
        ON authors.Event_Num = events.Event_ID
INNER JOIN authors
        ON papers.Author_Num = authors.Author_ID
WHERE events.Event_ID < 100;
```

**Database 3:**

```
SELECT
        Event_Name,
        presentations.Author_ID,
        Paper_Title,
        Presentation_Topic
FROM
        authors,
        papers
INNER JOIN events
        ON authors.Event_Num = events.Event_ID
INNER JOIN presentations
        ON papers.Author_Num = presentations.Author_ID
WHERE events.Event_ID < 100;
```

9. Test the time in ns for executing the 3 queries on each database, by running each of them 1000 times.

## QUERY 1:

### Database 1

```
ne $ cat task9_1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[150] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_1.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Chair_Num, Venue_Address, Event_Name FROM venues INNER JOIN events ON events.Event_ID = ve
nues.Event_Num");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
cshoop2018@code01 one $ time ./task9_1
3.7.17
Closing database...

real    0m3.551s
user    0m0.139s
sys     0m0.370s
cshoop2018@code01 one $
```

## Database 2

```
cshoop2018@code01 two $ gcc -o task9_1 task9_1.c -lsqlite3 && cat task9_1.c && time ./task9_1
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_2.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Chair_Num, Venue_Address, Event_Name FROM chairs INNER JOIN events ON events.Event_ID = ve
nues.Event_Num INNER JOIN venues ON venues.Venue_ID = chairs.Venue_ID");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m7.910s
user    0m3.103s
sys     0m0.611s
cshoop2018@code01 two $
```

## Database 3

```
cshoop2018@code01 three $ gcc -o task9_1 task9_1.c -lsqlite3 && cat task9_1.c && time ./task9_1
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_3.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Chair_Num, Venue_Address, Event_Name FROM chairs INNER JOIN events ON events.Event_ID = ve
nues.Event_Num INNER JOIN venues ON venues.Venue_ID = chairs.Venue_ID");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m3.656s
user    0m0.164s
sys     0m0.457s
cshoop2018@code01 three $
```

## QUERY 2:

### Database 1

```
cshoop2018@code01 one $ gcc -o task9_2 task9_2.c -lsqlite3 && cat task9_2.c && time ./task9_2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_1.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Reviewer_Lastname, Paper_Title, Presentation_Topic, Author_Firstname, Author_Lastname FROM
 reviewers INNER JOIN authors ON authors.Author_ID = papers.Author_Num INNER JOIN papers ON papers.Paper_ID = reviewer
s.Paper_Num");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m3.625s
user    0m0.151s
sys     0m0.389s
cshoop2018@code01 one $
```

## Database 2

```
cshoop2018@code01 two $ gcc -o task9_2 task9_2.c -lsqlite3 && cat task9_2.c && time ./task9_2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_2.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Reviewer_Lastname, Paper_Title, Presentation_Topic, Author_Firstname, Author_Lastname FROM
 reviewers INNER JOIN authors ON authors.Author_ID = papers.Author_Num INNER JOIN papers ON papers.Paper_ID = reviewer
s.Paper_Num");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m3.756s
user    0m0.211s
sys     0m0.378s
cshoop2018@code01 two $
```

## Database 3

```
cshoop2018@code01 three $ gcc -o task9_2 task9_2.c -lsqlite3 && cat task9_2.c && time ./task9_2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_3.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Reviewer_Lastname, Paper_Title, a1.Author_Name, Presentation_Topic, a2.Author_Name FROM re
viewers, presentations INNER JOIN authors a1 ON papers.Author_Num = a1.Author_ID INNER JOIN papers ON papers.Paper_ID
= reviewers.Paper_Num INNER JOIN authors a2 ON presentations.Author_ID = a2.Author_ID");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m12.144s
user    0m7.878s
sys     0m0.464s
cshoop2018@code01 three $
```

## QUERY 3:

### Database 1

```
cshoop2018@code01 one $ gcc -o task9_3 task9_3.c -lsqlite3 && cat task9_3.c && time ./task9_3
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_1.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Event_Name, Author_ID, Paper_Title, Presentation_Topic FROM papers INNER JOIN events ON au
thors.Event_Num = events.Event_ID INNER JOIN authors ON papers.Author_Num = authors.Author_ID WHERE events.Event_ID <
100;");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m4.404s
user    0m0.240s
sys     0m0.600s
cshoop2018@code01 one $
```

## Database 2

```
cshoop2018@code01 two $ gcc -o task9_3 task9_3.c -lsqlite3 && cat task9_3.c && time ./task9_3
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_2.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Event_Name, Author_ID, Paper_Title, Presentation_Topic FROM papers INNER JOIN events ON au
thors.Event_Num = events.Event_ID INNER JOIN authors ON papers.Author_Num = authors.Author_ID WHERE events.Event_ID <
100;");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m4.543s
user    0m0.395s
sys     0m0.638s
cshoop2018@code01 two $
```

## Database 3

```
cshoop2018@code01 three $ gcc -o task9_3 task9_3.c -lsqlite3 && cat task9_3.c && time ./task9_3
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlite3.h>

int main(void) {
    char sql[500] = "", *err_msg = 0;
    printf("%s\n", sqlite3_libversion());

    sqlite3 *db;

    int rc = sqlite3_open("shoop_meyer_project_3.sqlite", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return (1);
    }

    for (int i = 0; i < 1000; i++) {
        strcpy(sql, "SELECT Event_Name, presentations.Author_ID, Paper_Title, Presentation_Topic FROM authors, papers
INNER JOIN events ON authors.Event_Num = events.Event_ID INNER JOIN presentations ON papers.Author_Num = presentations
.Author_ID WHERE events.Event_ID < 100;");

        rc = sqlite3_exec(db, sql, 0, 0, &err_msg);
        if (rc != SQLITE_OK) {
            fprintf(stderr, "SQL error: %s\n", err_msg);
            sqlite3_free(err_msg);
            sqlite3_close(db);
            return (1);
        }
    }

    printf("Closing database...\n");
    sqlite3_close(db);

    return (0);
}
3.7.17
Closing database...

real    0m5.869s
user    0m0.956s
sys     0m0.659s
cshoop2018@code01 three $
```