



CS 353 – Database Systems

Scientific Papers Data Management  
System

**Project Design Report**  
Group 3

**Project URL:**

[gokberkkaraca.com/scientific-papers-database](http://gokberkkaraca.com/scientific-papers-database)

<b>Ali Sabbagh</b>	<b>21500269</b>
<b>Kaan Sancak</b>	<b>21502708</b>
<b>Kanan Asadov</b>	<b>21503382</b>
<b>Sabit Gökberk Karaca</b>	<b>21401862</b>

<b>Revised E/R Model</b>	<b>4</b>
<b>Relation Schemas</b>	<b>6</b>
Subscriber	6
Institution	7
Audience	8
Conference	9
Journal	10
Publisher	11
Author	12
Editor	13
Reviewer	14
Submission	15
Publication	16
Sponsor	17
Subscription	18
Invites	19
Expertise	20
AuthorExpertise	21
ReviewerExpertise	22
Reviews	23
EditorPublisher	24
Finances	25
Cites	26
Submits	27
<b>Functional Dependencies and Normalization of Tables</b>	<b>28</b>
<b>Functional Components</b>	<b>29</b>
Use Cases / Scenarios	29
Reviewer Use Cases	29
Editor Use Cases	30
Author Use Cases	31
Subscriber Use Cases	32
Audience Use Cases	33
<b>User Interface Design and Corresponding SQL Statements</b>	<b>34</b>
Sign-in Page	34
Sign-up Page	35
Journal Volume Page	37
Main Page	39
Invite Reviewer Page	41
Publication Page	43
Author Submissions Page	44

Author Publish Page	46
Conference Page	47
Reviewer Submissions Page	48
Write Feedback Page	49
Editor Submissions Page	50
Conference Audience Page	52
Feedback Popup	53
<b>Advanced Database Components</b>	<b>54</b>
Views	54
Main Page View	54
Author Submissions View	54
Editor Submissions View	54
Reviewer Submissions View	55
Stored Procedures	55
Author Signup	55
Editor Signup	55
Reviewer Signup	55
Publish Paper	56
Submit Paper	56
Get Authors of Submission	56
Reports	56
Average number of authors per publication for every publisher	57
Triggers	57
Constraints	57
<b>Implementation Plan</b>	<b>58</b>

# 1. Revised E/R Model

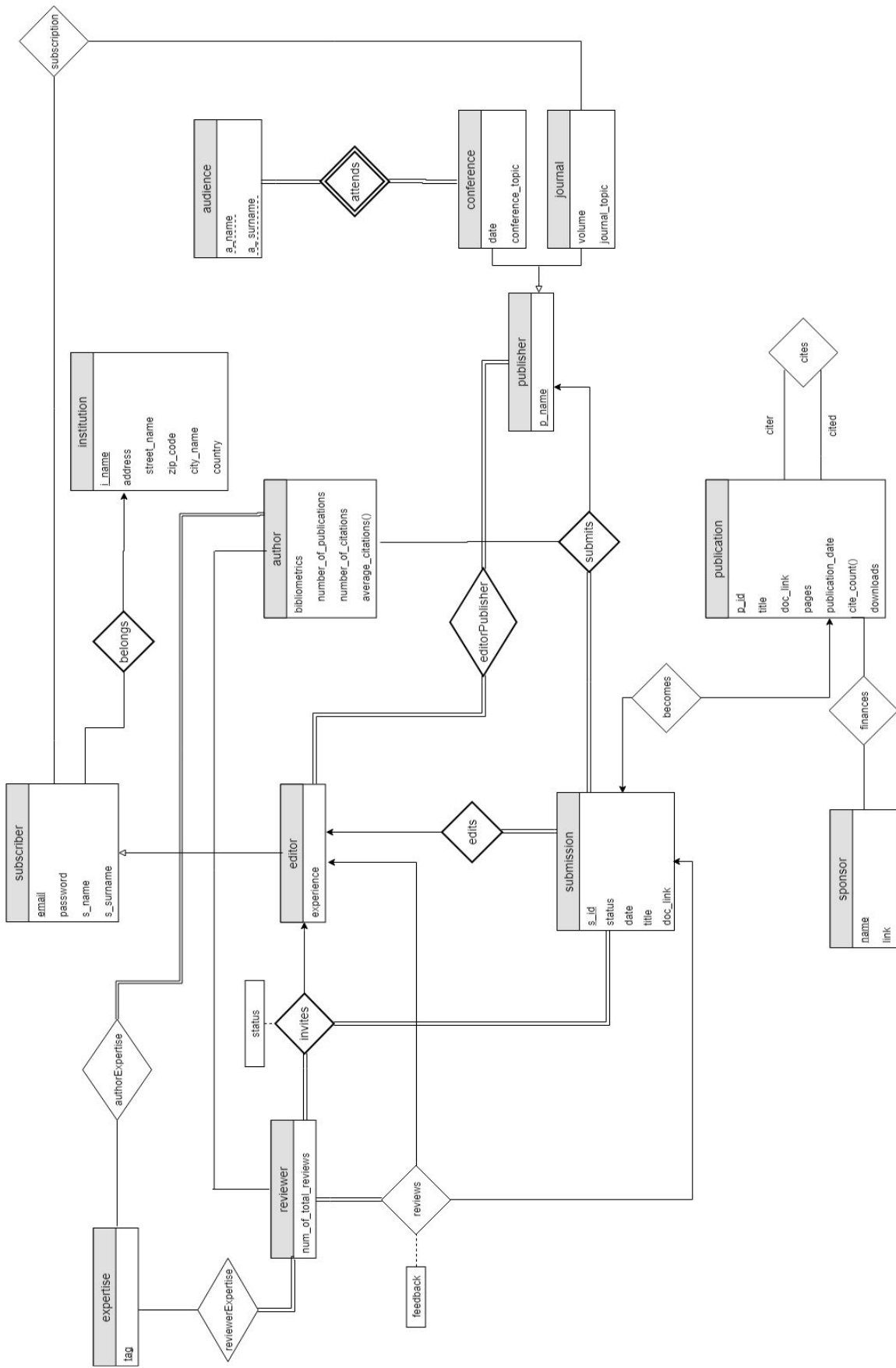
After our teaching assistant reviewed our proposal report, we received a feedback from the assistant and revised the E/R model according to the feedback. We have corrected the mistakes on E/R model and removed some redundant attributes or tables. The changes are as follows;

Changes to entities:

- We made publisher's name the primary key of the entity and converted conference's and journal's primary keys to normal attributes.
- We converted cite\_count attribute of publication to a derived attribute since it can be calculated using the cites table.
- We removed patent entity because it was irrelevant.
- We converted the audience entity to a weak entity since it is meaningless without conference.
- We removed topic attribute from submission and publication entities.
- We removed the publisher attribute of publication since it can be found from using the relationship between publication and submission.

Changes to relationships:

- We changed the cardinality of submission in edits relationship to many.
- We changed the cardinality of submission in submits relationship to many, we also changed the participation type to total participation.
- We added a new relation, called cites, to keep track of the citations between different publications.
- We removed has relationship between patent, author and publication because patent entity is removed.
- We changed the entity participation of conference in attends relationship to total participation.
- We removed cooperates relationship from the E/R model, which was between author and publication.
- We removed precedes relationship from the E/R diagram, it was irrelevant.
- We changed the entity participation of submission in invites relationship to total participation, we also changed the cardinality to many.



## 2. Relation Schemas

### 2.1. Subscriber

**Relational Model:**

subscriber (email, password, s\_name, s\_surname, ref\_institution\_name)

**Functional Dependencies:**

email -> password, s\_name, s\_surname, ref\_institution\_name

**Candidate Keys:**

{ (email) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE subscriber(  
    email varchar(200) PRIMARY KEY,  
    password varchar(50),  
    s_name varchar(50),  
    s_surname varchar(50),  
    ref_institution_name varchar(200),  
    FOREIGN KEY(ref_institution_name)  
    REFERENCES institution(i_name) )
```

## 2.2. Institution

### Relational Model:

Institution (i\_name, street\_name, zip\_code, city\_name, country)

### Functional Dependencies:

i\_name -> street\_name, zip\_code, city\_name, country

### Candidate Keys:

{ (i\_name) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE institution(  
    i_name varchar(200) PRIMARY KEY,  
    street_name varchar(50),  
    zip_code varchar(10),  
    city_name varchar(50),  
    country varchar(50) )
```

## 2.3. Audience

### Relational Model:

Audience (ref\_p\_name, a\_name, a\_surname)

### Functional Dependencies:

No dependencies

### Candidate Keys:

{ (ref\_p\_name, a\_name, a\_surname) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE audience(  
    ref_p_name varchar(50),  
    a_name varchar(50),  
    a_surname varchar(50),  
    PRIMARY KEY (ref_p_name, a_name, a_surname),  
    FOREIGN KEY (ref_p_name) REFERENCES  
conference(ref_publisher_name))
```



## 2.4. Conference

### Relational Model:

conference (date, cnf\_topic, ref\_publisher\_name)

### Functional Dependencies:

ref\_publisher\_name -> date, cnf\_topic

### Candidate Keys:

{ (ref\_publisher\_name) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE conference(  
    date date,  
    cnf_topic varchar(200),  
    ref_publisher_name varchar(200) NOT NULL,  
    FOREIGN KEY(ref_publisher_name)  
    REFERENCES publisher(p_name))
```

## 2.5. Journal

### Relational Model:

journal (volume, journal\_topic, ref\_publisher\_name)

### Functional Dependencies:

ref\_publisher\_name -> volume, journal\_topic

### Candidate Keys:

{ (ref\_publisher\_name) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE journal(  
    volume decimal(5,2),  
    journal_topic varchar(200),  
    ref_publisher_name varchar(200) NOT NULL,  
    FOREIGN KEY (ref_publisher_name) REFERENCES  
publisher(p_name))
```

## 2.6. Publisher

**Relational Model:**

publisher (p\_name)

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{ (p\_name) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE publisher( p_name varchar(200) PRIMARY KEY )
```

## 2.7. Author

### Relational Model:

author (ref\_subscriber\_email, num\_publications, num\_citations, avg\_citations\_paper)

### Functional Dependencies:

ref\_subscriber\_email -> num\_publications, num\_citations, avg\_citations\_paper

### Candidate Keys:

{ (ref\_subscriber\_email) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE author(  
    num_publications INTEGER,  
    num_citations INTEGER,  
    avg_citations_paper decimal(10,2) DEFAULT 0,  
    ref_subscriber_email varchar(200),  
    FOREIGN KEY (ref_subscriber_email) REFERENCES  
subscriber(email))
```

## 2.8. Editor

### Relational Model:

Editor (ref\_subscriber\_email, experience)

### Functional Dependencies:

ref\_subscriber\_email -> experience

### Candidate Keys:

{ (ref\_subscriber\_email) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE editor(  
    experience INTEGER,  
    ref_subscriber_email varchar(200),  
    FOREIGN KEY (ref_subscriber_email) REFERENCES  
subscriber(email))
```

## 2.9. Reviewer

### Relational Model:

reviewer (num\_total\_reviews, ref\_subscriber\_email)

### Functional Dependencies:

ref\_subscriber\_email -> num\_total\_reviews

### Candidate Keys:

{ (ref\_subscriber\_email) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE reviewer(  
    num_total_reviews INT DEFAULT 0,  
    ref_subscriber_email varchar(200) NOT NULL,  
    FOREIGN KEY (ref_subscriber_email) REFERENCES  
subscriber(email))
```

## 2.10. Submission

### Relational Model:

submission (s\_id, status, title, ref\_editor\_email)

### Functional Dependencies:

s\_id -> status, title, ref\_editor\_email

### Candidate Keys:

{ (s\_id) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE submission(  
    s_id INT PRIMARY KEY AUTO_INCREMENT,  
    status TINYINT,  
    title varchar(500),  
    doc_link varchar(200),  
    date date,  
    ref_editor_email varchar(200) NOT NULL,  
    FOREIGN KEY (ref_editor_email) REFERENCES  
editor(ref_subscriber_email) )
```

Status can be:

- 0 submitted by authors
- 1 assigned to reviewers by editor
- 2 editor last checking, with reviewers feedback
- 3 sent to author(s) to approve changes
- 4 changes accepted/ paper sent to publication
- 5 submission rejected

## 2.11. Publication

### Relational Model:

publication (p\_id, title, pages, publication\_date, doc\_link, downloads, ref\_submission\_id)

### Functional Dependencies:

p\_id -> title, pages, topic, publication\_date, downloads, ref\_submission\_id

### Candidate Keys:

{ (p\_id) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE publication(  
    p_id INT PRIMARY KEY AUTO_INCREMENT,  
    Title varchar(500),  
    pages INT,  
    publication_date date,  
    doc_link varchar(200),  
    downloads INT DEFAULT 0,  
    ref_submission_id INT NOT NULL,  
    FOREIGN KEY (ref_submission_id)  
    REFERENCES submission(s_id) )
```



## 2.12. Sponsor

**Relational Model:**

sponsor (name, link)

**Functional Dependencies:**

Name-> link

**Candidate Keys:**

{ (name) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE sponsor(  
    name varchar(200) PRIMARY KEY,  
    link varchar(200) )
```

## 2.13. Subscription

**Relational Model:**

subscription ( ref\_subscriber\_email, ref\_journal )

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{{ (ref\_subscriber\_email, ref\_journal) }}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE subscription(  
    ref_subscriber_email varchar(200) NOT NULL,  
    FOREIGN KEY (ref_subscriber_email)  
    REFERENCES subscriber(email),  
    ref_journal varchar(200) NOT NULL,  
    FOREIGN KEY (ref_journal)  
    REFERENCES journal(ref_publisher_name) )
```

## 2.14. Invites

### Relational Model:

invites ( ref\_reviewer\_email, ref\_editor\_email, ref\_submission\_id, status)

### Functional Dependencies:

ref\_reviewer\_email, ref\_editor\_email, ref\_submission\_id -> status

### Candidate Keys:

{{ ref\_reviewer\_email, ref\_editor\_email, ref\_submission\_id }}

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE invites(  
    ref_reviewer_email varchar(200) NOT NULL,  
    ref_editor_email varchar(200) NOT NULL,  
    ref_submission_id INT NOT NULL,  
    FOREIGN KEY (ref_editor_email)  
    REFERENCES editor(ref_subscriber_email),  
    FOREIGN KEY (ref_reviewer_email)  
    REFERENCES reviewer(ref_subscriber_email),  
    FOREIGN KEY (ref_submission_id)  
    REFERENCES submission(s_id),  
    status TINYINT DEFAULT 0)
```

## 2.15. Expertise

**Relational Model:**

expertise (tag)

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{{ tag }}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE expertise( tag varchar(100) PRIMARY KEY)
```

## 2.16. AuthorExpertise

### Relational Model:

authorExpertise( ref\_author\_email, ref\_tag )

### Functional Dependencies:

No dependencies

### Candidate Keys:

{( ref\_author\_email, ref\_tag )}

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE authorExpertise(  
    ref_author_email varchar(200) NOT NULL,  
    ref_tag varchar(100) NOT NULL,  
    FOREIGN KEY (ref_author_email)  
    REFERENCES author(ref_subscriber_email),  
    FOREIGN KEY (ref_tag) REFERENCES expertise(tag) )
```

## 2.17. ReviewerExpertise

### Relational Model:

reviewerExpertise ( ref\_reviewer\_email, ref\_tag )

### Functional Dependencies:

No dependencies

### Candidate Keys:

{{ ref\_reviewer\_email, ref\_tag }}

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE reviewerExpertise (  
    ref_reviewer_email varchar(200) NOT NULL,  
    ref_tag varchar(100) NOT NULL,  
    FOREIGN KEY (ref_reviewer_email)  
    REFERENCES reviewer(ref_subscriber_email),  
    FOREIGN KEY (ref_tag) REFERENCES expertise(tag) )
```

## 2.18. Reviews

### Relational Model:

reviews ( ref\_reviewer\_email, ref\_editor\_email, ref\_submission\_id, feedback)

### Functional Dependencies:

ref\_reviewer\_email, ref\_editor\_email, ref\_submission\_id -> feedback

### Candidate Keys:

{{ ref\_reviewer\_email, ref\_editor\_email, ref\_submission\_id }}

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE reviews(  
    ref_reviewer_email varchar(200) NOT NULL,  
    ref_editor_email varchar(200) NOT NULL,  
    ref_submission_id INT NOT NULL,  
    feedback varchar(2500),  
    FOREIGN KEY (ref_reviewer_email)  
    REFERENCES reviewer(ref_subscriber_email),  
    FOREIGN KEY (ref_editor_email)  
    REFERENCES editor(ref_subscriber_email),  
    FOREIGN KEY (ref_submission_id)  
    REFERENCES submission(s_id))
```

## 2.19. EditorPublisher

### Relational Model:

editorPublisher ( ref\_editor\_email, ref\_publisher\_name )

### Functional Dependencies:

No dependencies

### Candidate Keys:

{{ ref\_editor\_email, ref\_publisher\_name }}

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE editorPublisher (  
    ref_editor_email varchar(200) NOT NULL,  
    ref_publisher_name varchar(200) NOT NULL,  
    FOREIGN KEY (ref_editor_email)  
    REFERENCES editor(ref_subscriber_email),  
    FOREIGN KEY (ref_publisher_name)  
    REFERENCES publisher(p_name))
```



## 2.20. Finances

### Relational Model:

finances ( ref\_sponsor\_name, ref\_publication\_id )

### Functional Dependencies:

No dependencies

### Candidate Keys:

{( ref\_sponsor\_name, ref\_publication\_id )}

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE finances(  
    ref_sponsor_name varchar(200) NOT NULL,  
    ref_publication_id INT NOT NULL,  
    FOREIGN KEY (ref_sponsor_name) REFERENCES sponsor(name),  
    FOREIGN KEY (ref_publication_id)  
    REFERENCES publication(p_id) )
```

## 2.21. Cites

**Relational Model:**

cites (citer\_id, cited\_id)

**Functional Dependencies:**

No dependencies

**Candidate Keys:**

{{ citer\_id, cited\_id }}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE cites(  
    citer_id INT NOT NULL,  
    cited_id INT NOT NULL,  
    FOREIGN KEY (citer_id) REFERENCES publication(p_id),  
    FOREIGN KEY (cited_id) REFERENCES publication(p_id) )
```

## 2.22. Submits

### Relational Model:

submits ( ref\_author\_email, ref\_submission\_id, ref\_publisher\_name )

### Functional Dependencies:

No dependencies

### Candidate Keys:

{ ( ref\_author\_email, ref\_submission\_id, ref\_publisher\_name ) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE submits (
    ref_author_email varchar(200) NOT NULL,
    ref_submission_id INT NOT NULL,
    ref_publisher_name varchar(200) NOT NULL,
    FOREIGN KEY (ref_author_email)
        REFERENCES author(ref_subscriber_email),
    FOREIGN KEY (ref_submission_id)
        REFERENCES submission(s_id),
    FOREIGN KEY (ref_publisher_name)
        REFERENCES publisher(p_name) )
```

### 3. Functional Dependencies and Normalization of Tables

As we followed good design principles, and with the help of proposal report notes from TA, we were able to design our database relations to be in Boyce-Codd form. There is no data redundancy and no decomposition is required. Normal form is specified for each relation in Relational Schemas (section 2).

## 4. Functional Components

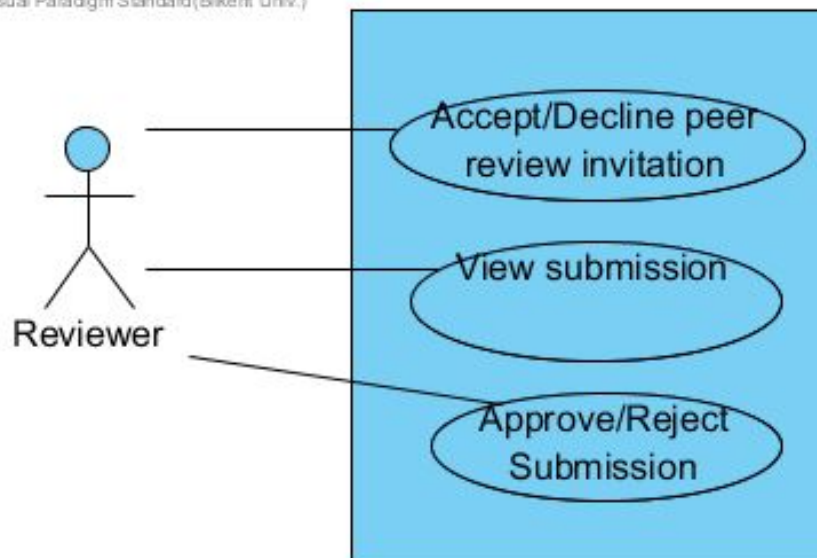
### 4.1. Use Cases / Scenarios

In scientific paper data management system, there are 6 unique types of users/actors which are reviewer, editor, author, subscriber, sponsor, and audience. In the rest of this section, the details of these actors are given. Some use cases such as signup, login, view/search publication are left out for clearness because these use cases are common to all actors. The system will provide different functionalities and limited features according to actor's type.

#### 4.1.1. Reviewer Use Cases

- Reviewers can login the system with e-mail address and password.
- Reviewers can accept peer review invitations from editors.
- Reviewers can reject peer review invitations from editors.
- Reviewers can view the submission if they have accepted the peer review invitation.
- Reviewers can approve the submission by sending a report to the editor.
- Reviewers can reject the submission by sending a report to the editor.
- Reviewers can search for publications, institutions, journals and conferences.
- Reviewers can view details of publications, institutions, journals and conferences.

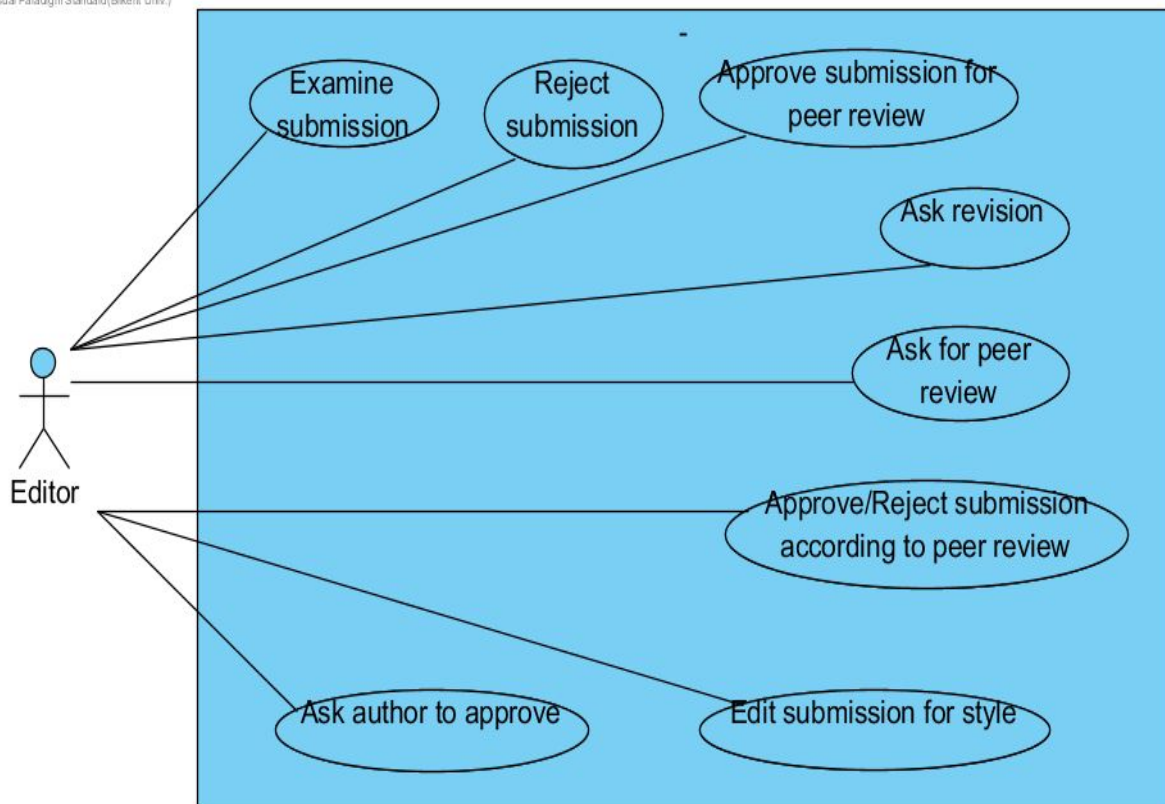
Visual Paradigm Standard (Bilkent Univ.)



### 4.1.2. Editor Use Cases

- Editors can login the system with e-mail address and password.
- Editors can examine submissions.
- Editors can reject submissions if the submission is unsuitable for the journal.
- Editors can ask for revision of the submission if the submission can be made suitable.
- Editors can approve the requirements
- Editors can ask for peer review from more than one reviewers if the submission's requirements are approved.
- Editors can search for publications, institutions, journals and conferences.
- Editors can view details of publications, institutions, journals and conferences.
- Editors can approve to publish the submission according to peer reviewers' report.
- Editors can reject to publish the submission according to peer reviewers' report.
- Editors can edit the submission for style (citations, references, tables, clarity, grammar etc.) before sending it to publication.
- Editors can ask authors to approve final submission.

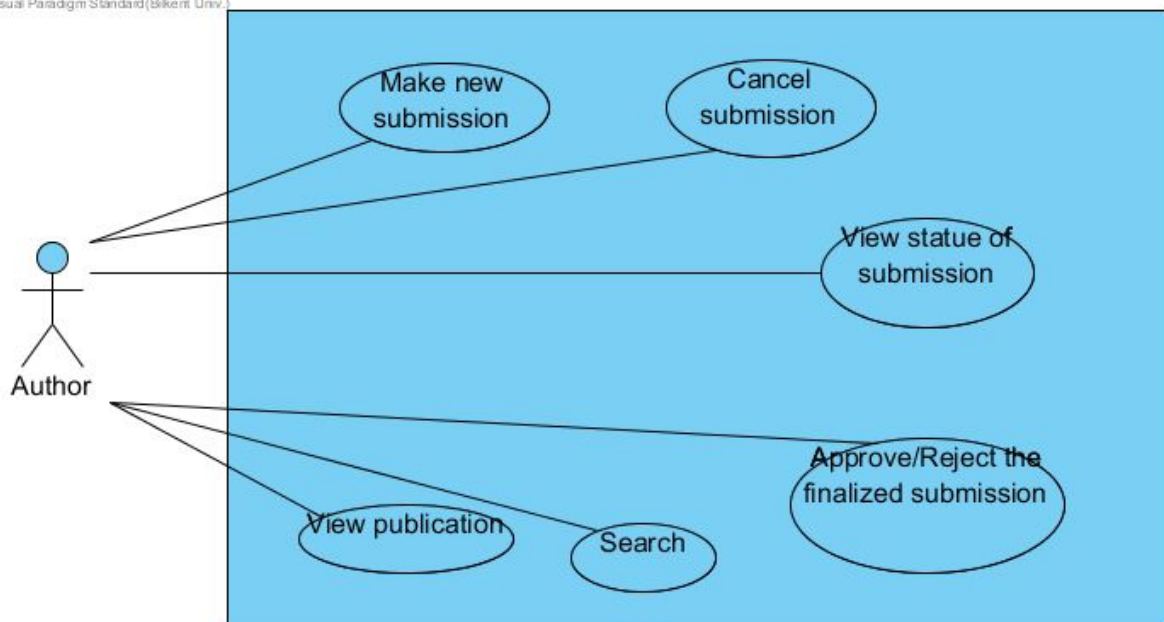
Visual Paradigm Standard (Bilkent Univ.)



### 4.1.3. Author Use Cases

- Authors can login the system with e-mail address and password.
- Authors can make new submissions to journals and conferences.
- Authors can cancel their submissions.
- Authors can view the status of their submissions.
- Authors can search for publications, institutions, journals and conferences.
- Authors can view details of publications, institutions, journals and conferences.
- Authors can approve the finalized submission.
- Authors can reject the finalized submission

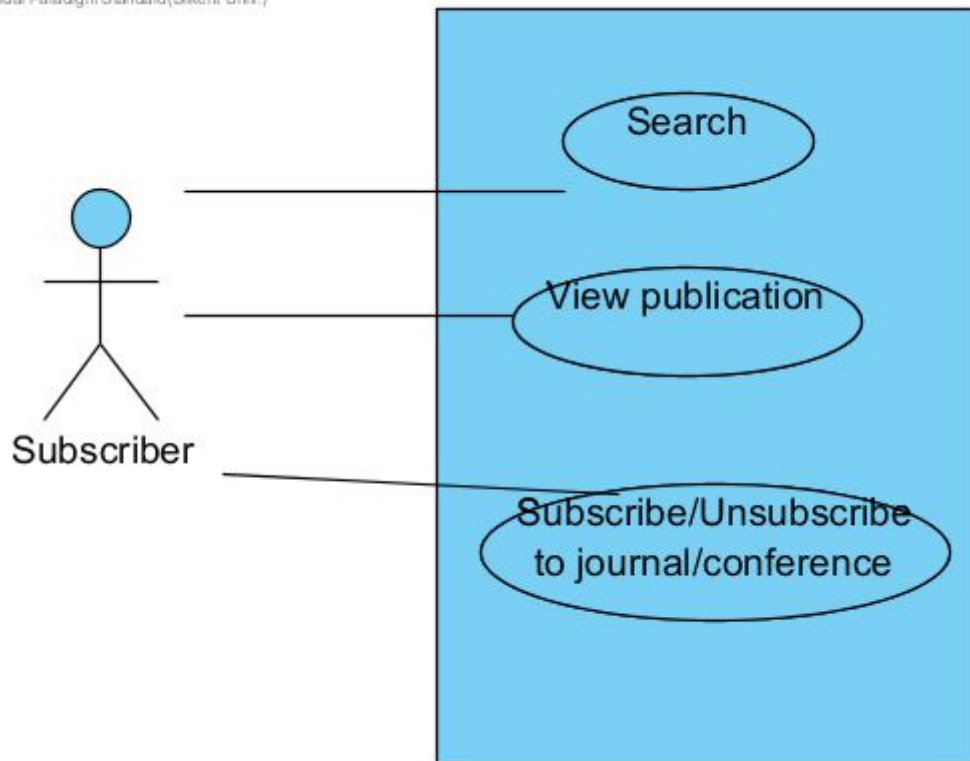
Visual Paradigm Standard (Bilkent Univ.)



#### 4.1.4. Subscriber Use Cases

- Subscribers can login the system with e-mail address and password.
- Subscribers can search for publications, institutions, journals and conferences.
- Subscribers can view details of publications, institutions, journals and conferences.
- Subscribers can subscribe/unsubscribe journals and conferences.

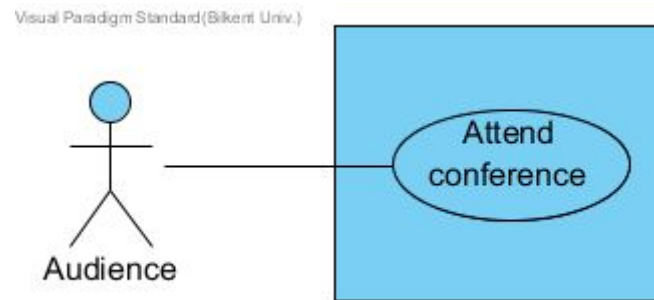
Visual Paradigm Standard (Bilkent Univ.)





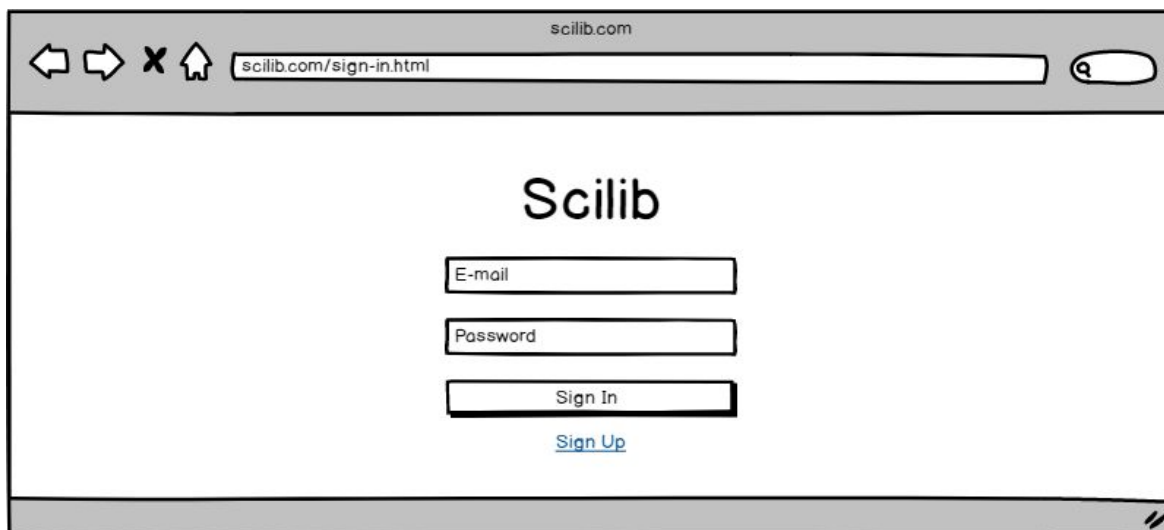
#### 4.1.5. Audience Use Cases

- Audience can attend conferences.



## 5. User Interface Design and Corresponding SQL Statements

### 5.1. Sign-in Page



A hand-drawn style mockup of a web browser window. The address bar shows 'scilib.com' and the URL 'scilib.com/sign-in.html'. The page content features the 'Scilib' logo at the top center. Below the logo are three input fields: 'E-mail', 'Password', and a 'Sign In' button. A 'Sign Up' link is positioned below the 'Sign In' button. The browser window includes standard navigation icons (back, forward, stop, home) and a search icon in the address bar.

**Inputs:** @email, @password

**Process:** The user enters its e-mail address and password to sign-in to the Scientific Papers Data Management System.

**SQL Statements:**

**Signing in:**

```
SELECT email
FROM subscriber
WHERE email = @email AND password = @password;
```

## 5.2. Sign-up Page

The image shows a web browser window with the URL `scilib.com/sign-up.html`. The page title is "Scilib". The form contains the following elements:

- Text input fields for "Name", "Surname", "E-mail", and "Password".
- A "Select Institution" dropdown menu with three placeholder options.
- A "Select Role" dropdown menu with four options: "Subscriber", "Reviewer", "Author", and "Editor".
- A "Select Field of Expertise" dropdown menu with three placeholder options. It is connected to the "Subscriber" role option in the "Select Role" menu by a plus sign (+).
- A "Select Publisher" dropdown menu with three placeholder options. It is connected to the "Author" and "Editor" role options in the "Select Role" menu by arrows.
- A "Sign Up" button.
- A "Sign In" link.

**Inputs:** @email, @name, @surname, @password, @role, @field\_of\_expertise\_table, @publisher, @institution

**Process:** The subscriber(user) enters its name, surname, e-mail and password to sign-up to the Scientific Papers Data Management System.

## **SQL Statements:**

### **Subscriber Signup:**

```
INSERT INTO subscriber VALUES (@email, @password, @s_name, @s_surname,  
@institution)
```

### **Author Signup:**

```
CALL authorSignUp( @email, @password, @s_name, @s_surname,  
@institution)
```

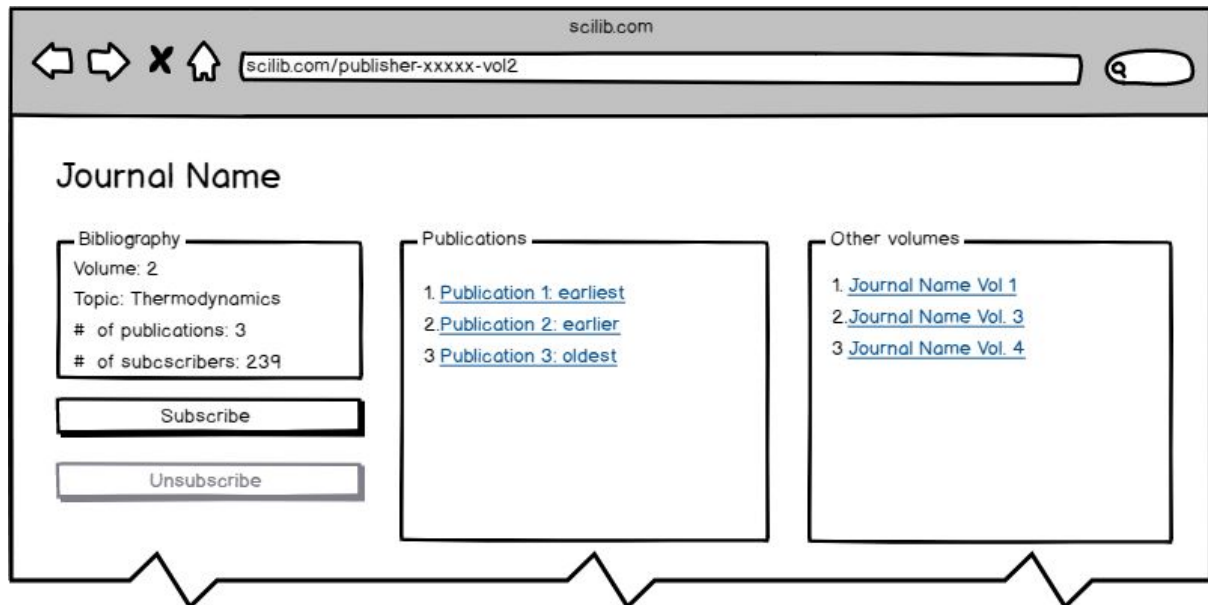
### **Editor Signup:**

```
CALL editorSignUp( @email, @password, @s_name, @s_surname,  
@institution)
```

### **Reviwer Signup:**

```
CALL signUp( @email, @password, @s_name, @s_surname, @institution)
```

### 5.3. Journal Volume Page



**Inputs:** @publisher\_name, @journal\_volume @user\_email

**Process:** To arrive to this page, the user should have clicked on the journal volume link. The link provides the @publisher\_name which is our primary key to look up this journal from database and volume number. The server return info about the journal volume including: topic, number of publications and subscribers, list of publications, and links to journal's other volumes.

Clicking a publication link: takes you to Publication view.

Clicking volume link: takes you to Journal Volume View again.

Subscribe/Unsubscribe: adds/removes the user to/from journal subscription.

#### SQL Statements:

##### Getting subscribers count:

```
SELECT count(*)
FROM subscription
WHERE ref_journal = @publisher_name
```

##### Getting publications count:

```
SELECT count(*)
FROM publication, journal, submits
WHERE publication.ref_submission_id = submits.ref_submission_id
AND journal.ref_publisher_name = submits.ref_publisher_name
AND volume = @journal_volume
```

**Displaying Journal Volume Publications:**

```
SELECT p_id, title
FROM publication, journal, submits
WHERE publication.ref_submission_id = submits.ref_submission_id
AND journal.ref_publisher_name = submits.ref_publisher_name
AND journal.ref_publisher_name = @publisher_name
AND volume = @journal_volume
ORDER BY publication_date desc
```

**Displaying Journal Other Volumes:**

```
SELECT volume
FROM journal
WHERE ref_publisher_name = @publisher_name
ORDER BY volume asc
```

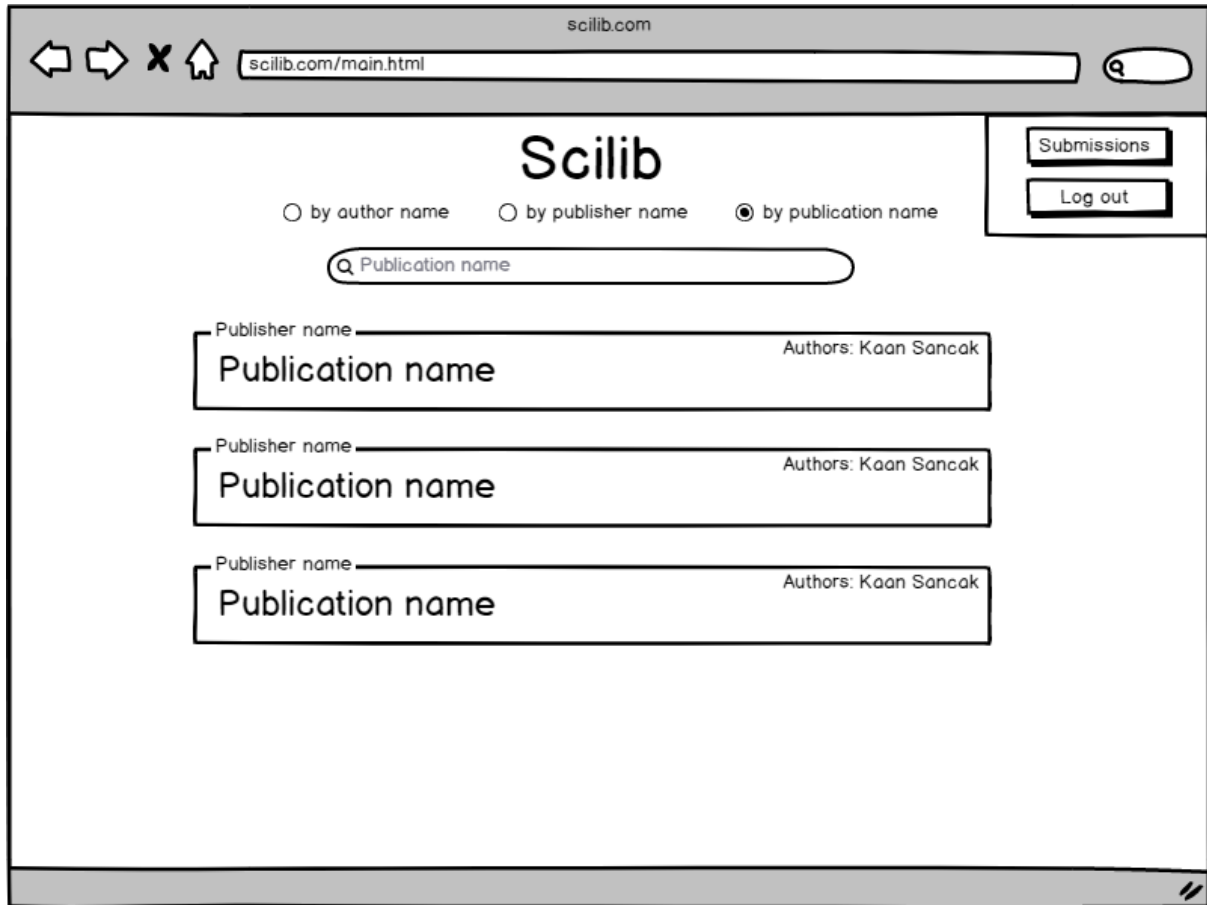
**Subscribing:**

```
INSERT INTO subscription values( @user_email, @publisher_name)
```

**Unsubscribe:**

```
DELETE FROM subscription
WHERE ref_subscriber_email = @user_email
AND ref_journal = @publisher_name
```

## 5.4. Main Page



The image shows a web browser window with the URL `scilib.com/main.html`. The page title is "Scilib". Below the title, there are three radio buttons for search type: "by author name", "by publisher name", and "by publication name". The "by publication name" option is selected. Below the radio buttons is a search input field with a magnifying glass icon and the placeholder text "Publication name". To the right of the search input field, there are two buttons: "Submissions" and "Log out". Below the search input field, there are three identical result boxes. Each box contains a "Publisher name" label, a "Publication name" input field, and an "Authors: Kaan Sancak" label. The browser window has a standard address bar with back, forward, and home buttons.

**Inputs:** @search\_type @search\_text

**Process:** A user can search publications in the database. User needs to select the search type first, application can search authors, publishers and publications. Search types will be indexed: author(0), publisher(1), publication(2). User enters the text in the search box and clicks the search button (Enter), database management system returns the matched results.

The server returns a list of publications, each has: the publisher name, the publication title and the names of the authors.

Clicking a publication link: takes you to Publication view.

Clicking a publisher link: takes you to Publisher View.

## SQL Statements:

### Searching publications:

```
SELECT *
FROM main_page_view
WHERE (CASE
    WHEN @searchtype = 0 then B.s_name like '@search_text%'
    WHEN @searchtype = 1 then S.ref_publisher_name like
'@search_text%'
    WHEN @searchtype = 2 then P.title like '@search_text%'
END)
```



## 5.5. Invite Reviewer Page

**Inputs:** @name, @field\_of\_expertise, @selected\_reviewer\_email, @editor\_email, @submission\_id

**Process:** In order to add a new reviewer to a submission, editor presses the Invite Reviewer button in the Editor Submissions View. Editor can enter a name or choose a field of expertise or both to search reviewers. The server returns the reviewers who are eligible to review the paper and editor chooses one of them to send the invitation.

## SQL Statements:

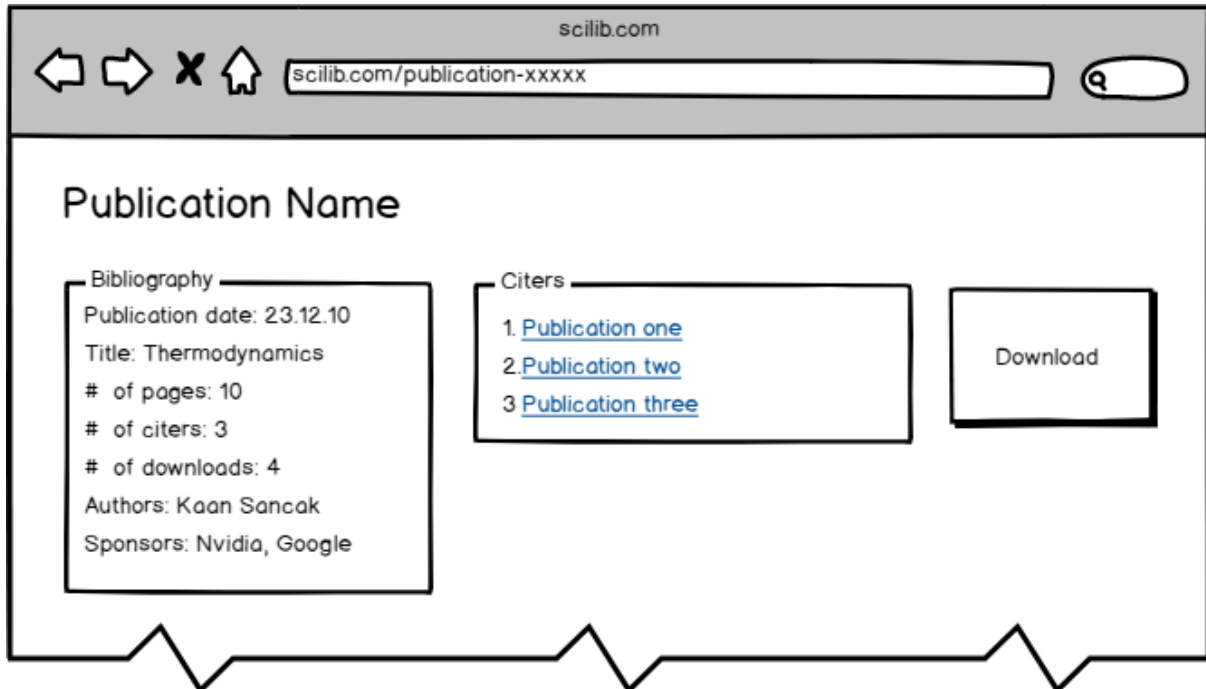
### Searching Reviewers:

```
SELECT S.email, S.s_name, S.s_surname
FROM reviewer R, reviewerExpertise E, subscriber S
WHERE R.ref_subscriber_email = E.ref_reviewer_email
      AND S.email = R.ref_subscriber_email
      AND E.ref_tag = @field_of_expertiese
      AND (CASE
            WHEN LENGTH(@name) = 0 THEN 1
            ELSE S.s_name = @name
          )
END)
```

### Inviting Reviewers

```
INSERT INTO
    invites (ref_reviewer_email, ref_editor_email,
ref_submission_id)
VALUES( @selected_reviewer_email, @editor_email,
@submission_id)
```

## 5.6. Publication Page



**Inputs:** @publication\_id

**Process:** User gets to this page after clicking a publication link. The server returns info of the publication, links to publications that cited this publication.

### SQL Statements:

#### Getting publication info:

```
SELECT publication_date, title, pages, downloads
FROM publication
WHERE p_id = @publication_id
```

#### Getting citers count

```
SELECT count( * )
FROM citation
WHERE cited_id = @publication_id
```

#### Getting citers info:

```
SELECT p_id, title
FROM publication
WHERE p_id in (SELECT citer_id from citation WHERE cited_id =
@publication_id)
```

## 5.7. Author Submissions Page

Title	Publisher	Status	Feedback	Publish/Cancel
<a href="#">My frist submission</a>	Sci-Fi Journal	Rejcected	<a href="#">See feedback</a>	<a href="#">Publish</a> <a href="#">Cancel</a>
<a href="#">My second submission</a>	Sci-Fi Journal	Waiting for approval	<a href="#">See feedback</a>	<a href="#">Publish</a> <a href="#">Cancel</a>
<a href="#">My third submission</a>	Sci-Fi Journal	Approved	<a href="#">See feedback</a>	<a href="#">Publish</a> <a href="#">Cancel</a>
<a href="#">My fourth submission</a>	Sci-Fi Journal	Published	<a href="#">See feedback</a>	<a href="#">Publish</a> <a href="#">Cancel</a>

Field(s) of expertise

**Inputs:** @user\_email @submission\_id @new\_title, @publisher\_name  
@title\_to\_publish @pages @new\_editor

**@pages:** when the finalized version is submitted to publication, our server will count the number of pages and pass it into the insertion query to database.

**@new\_editor:** when the paper is submitted to publication, the server checks for an editor from the selected publisher and assigns the submission to him to edit.

**Process:** The user gets here if he/she is an author and pressed "Submissions" button in main page. The server returns data containing, the author submissions: it can be in one of the states described in the submission table schema. The author can see feedback on submissions and approve them. When paper is approved by editor, the author can ask for it to be published providing the sponsors if any. Author can add a new submission providing the needed info.

## **SQL Statements:**

### **Getting submissions info:**

```
SELECT *  
FROM author_submissions_view  
ORDER BY s_id desc
```

### **Getting a submission feedback:**

```
SELECT feedback  
FROM reviews  
WHERE ref_submission_id = @submission_id
```

### **Make submission:**

```
BEGIN TRANSACTION  
INSERT INTO submission (status,title,ref_editor_email)  
values(0,@new_title, @new_editor)  
submit_paper ( @new_title, @new_editor)  
END
```

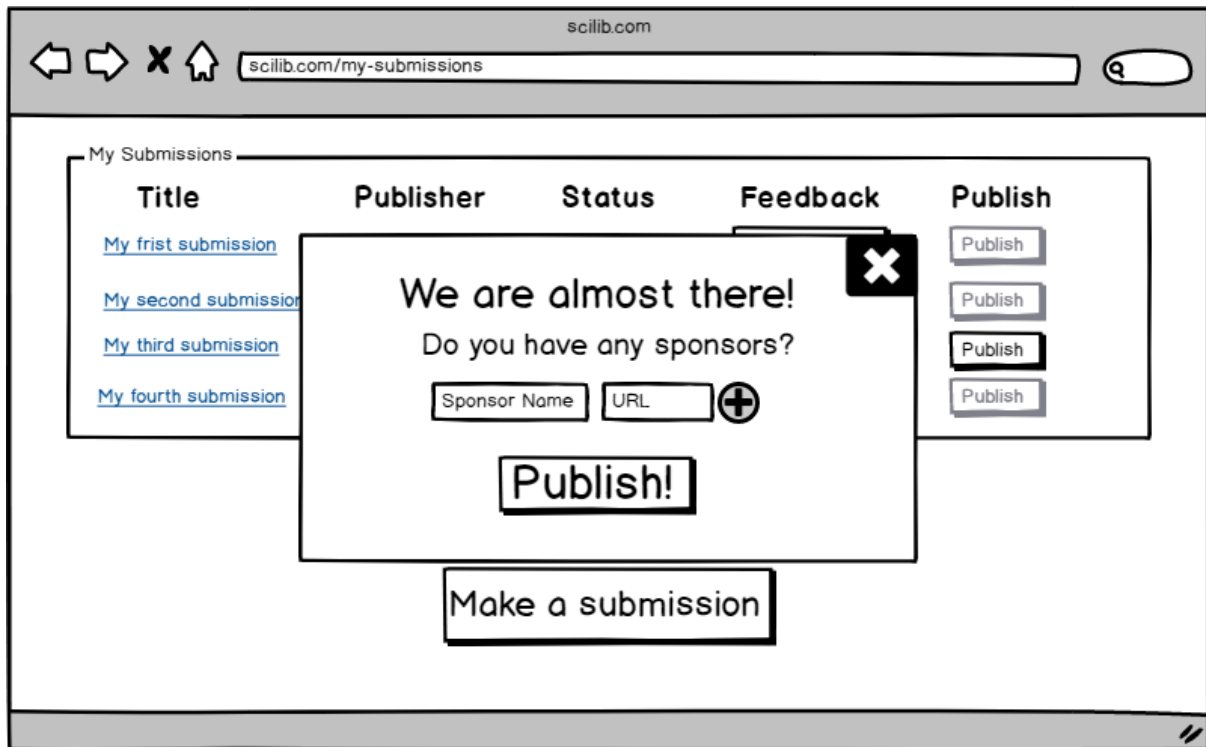
### **Publishing:**

```
publish_paper( @title_to_publish, @pages, @submission_id )
```

### **Cancelling:**

```
DELETE FROM submission  
WHERE s_id = @submission_id
```

## 5.8. Author Publish Page



**Inputs:** @sponsor\_names, @urls

**Process:** This pop up appears on the previous page. It's a continuation to the publication process.

**SQL Statments:**

**Adding sponsors to publication:**

```
Call addSponsorsToPublication( @sponsor_names, @urls)
```

## 5.9. Conference Page



**Inputs:** @publisher\_name

**Process:** To arrive to this page, the user should have clicked on the conference link. The link provides the @publisher\_name which is our primary key to look up this conference from database. The server returns info about the conference including: topic, number of publications, list of publications and the date the conference was held on.

Clicking a publication link: takes you to Publication view.

### SQL Statements:

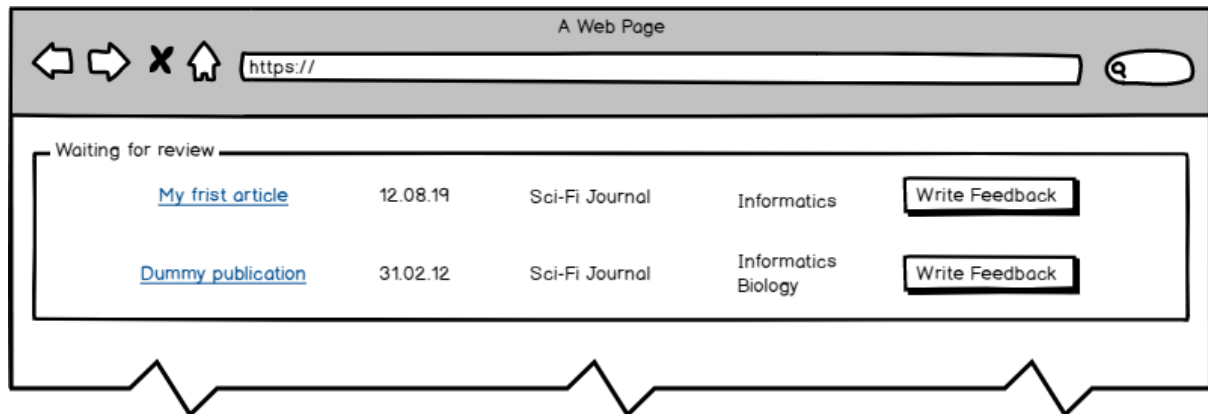
#### Getting publications count:

```
SELECT count(*)
FROM publication, conference, submits
WHERE publication.ref_submission_id = submits.ref_submission_id
AND conference.ref_publisher_name = submits.ref_publisher_name
```

#### Displaying Conference Publications:

```
SELECT p_id, title
FROM publication, conference, submits
WHERE publication.ref_submission_id = submits.ref_submission_id
AND conference.ref_publisher_name = submits.ref_publisher_name
AND conference.ref_publisher_name = @publisher_name
ORDER BY publication_date desc
```

## 5.10. Reviewer Submissions Page



**Inputs:** @reviewer\_email

**Process:** A reviewer can come to this page by clicking on the “Submissions” button on the main page. Reviewer’s email address is passed to the page and used for querying the database. Server returns the submissions that are assigned to the reviewer. Submission title, publisher name and the topic is displayed on this page.

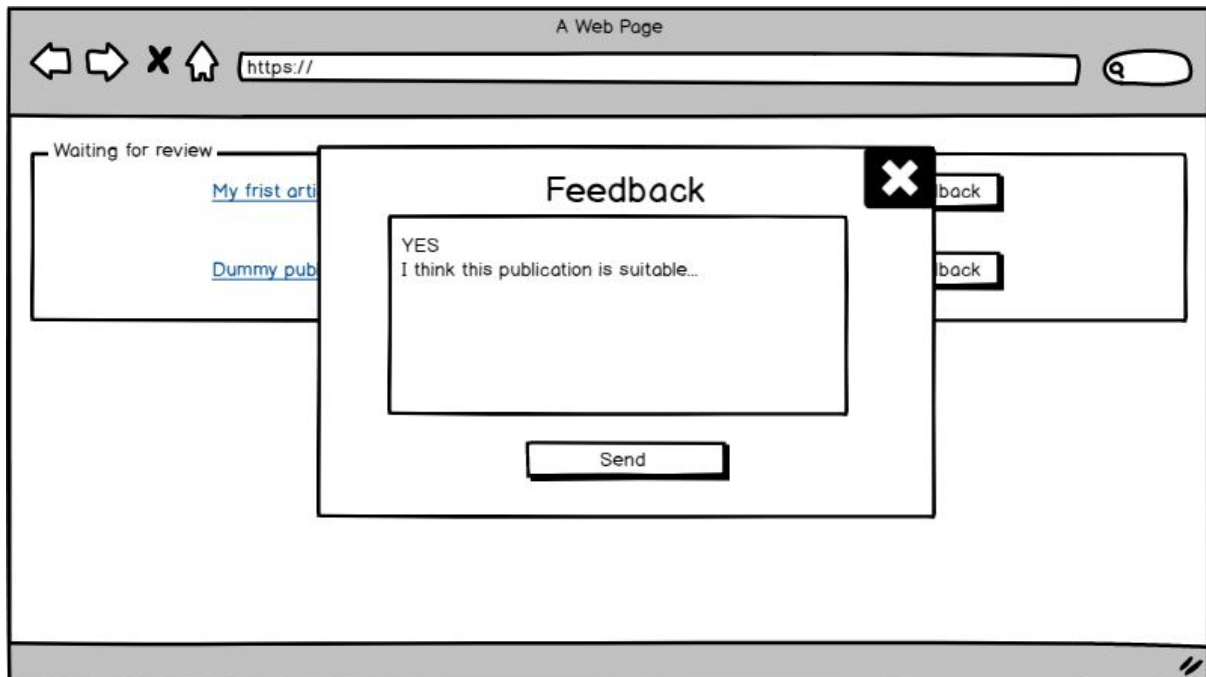
**SQL Statements:**

**Displaying Submissions:**

```
SELECT *  
FROM reviewer_submissions_view  
ORDER BY S.date DESC
```



## 5.11. Write Feedback Page



**Inputs:** @feedbacktext, @ref\_reviewer\_email, @submission\_id

**Process:** To arrive to this page, the reviewer should have selected the submission from submissions view. The link between two views provides @ref\_reviewer\_email and @submission\_id parameters. @submission\_id refers to the primary key of the submission and @ref\_reviewer\_email refers to the email of the reviewer. By clicking feedback button, reviewer can write feedback into the textbox and send it. The text is taken as @feedbacktext. By using the previous two parameters and @feedbacktext reviews table should be updated.

Clicking feedback: opens Feedback pop-up view.

Send: Adds the feedback to the submission which is under review

### SQL Statements:

#### Sending feedbacks:

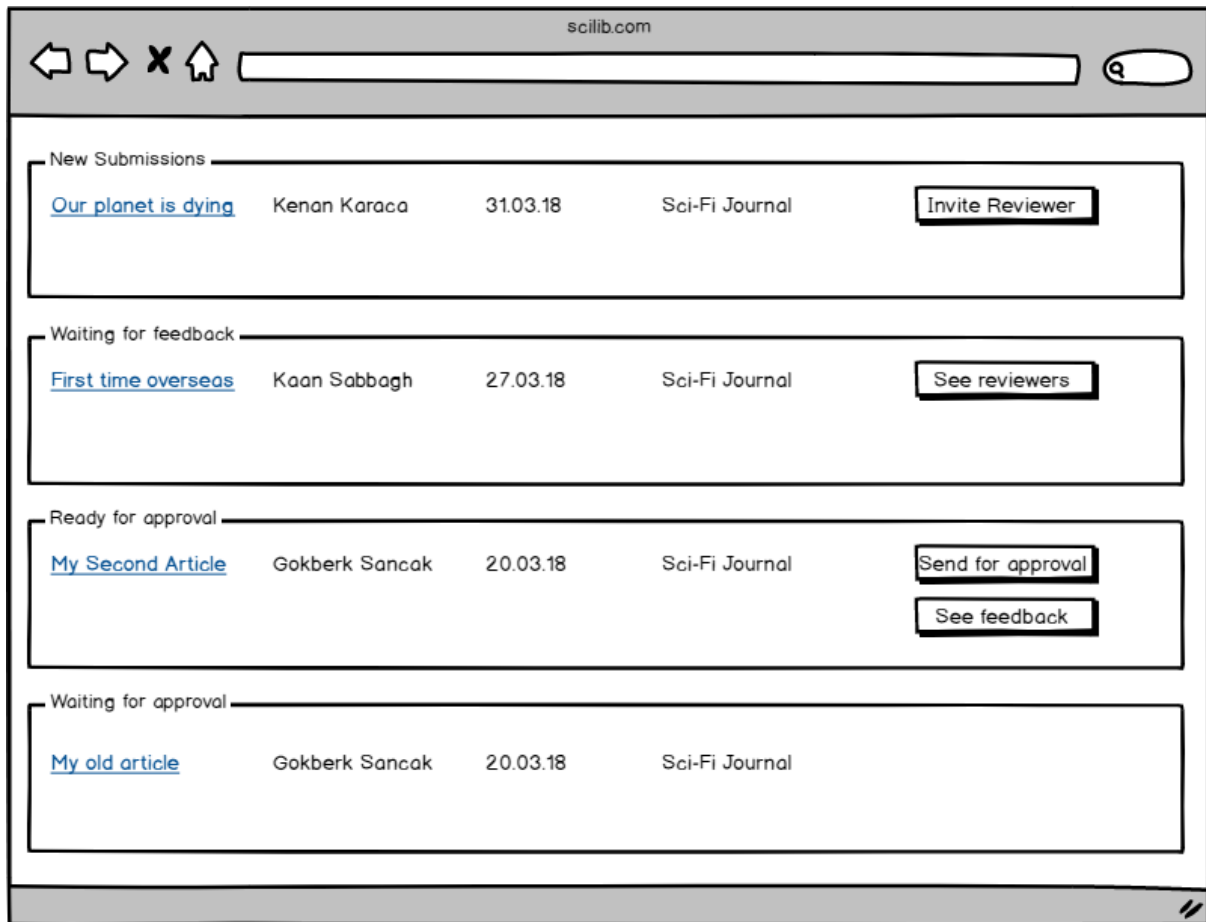
UPDATE reviews

SET feedback = @feedbacktext

WHERE reviews.ref\_reviewer\_email = @ref\_reviewer\_email

AND reviews.submission\_id = @submission\_id

## 5.12. Editor Submissions Page



scilib.com				
New Submissions				
<a href="#">Our planet is dying</a>	Kenan Karaca	31.03.18	Sci-Fi Journal	<button>Invite Reviewer</button>
Waiting for feedback				
<a href="#">First time overseas</a>	Kaan Sabbagh	27.03.18	Sci-Fi Journal	<button>See reviewers</button>
Ready for approval				
<a href="#">My Second Article</a>	Gokberk Sancak	20.03.18	Sci-Fi Journal	<button>Send for approval</button> <button>See feedback</button>
Waiting for approval				
<a href="#">My old article</a>	Gokberk Sancak	20.03.18	Sci-Fi Journal	

**Inputs:** @user\_email, @submission\_id

**Process:** To arrive to this page, the user who is an editor would have clicked on “Submissions” button from main page. The server returns data about the submissions that this editor is responsible for.

### SQL Statements:

#### Getting new submissions:

```
SELECT *  
FROM editor_submissions_view  
ORDER BY date ASC
```

For other types of submissions: waiting feedback, ready for approval and waiting approval.. The status number changes in sql as described in submission table schema.

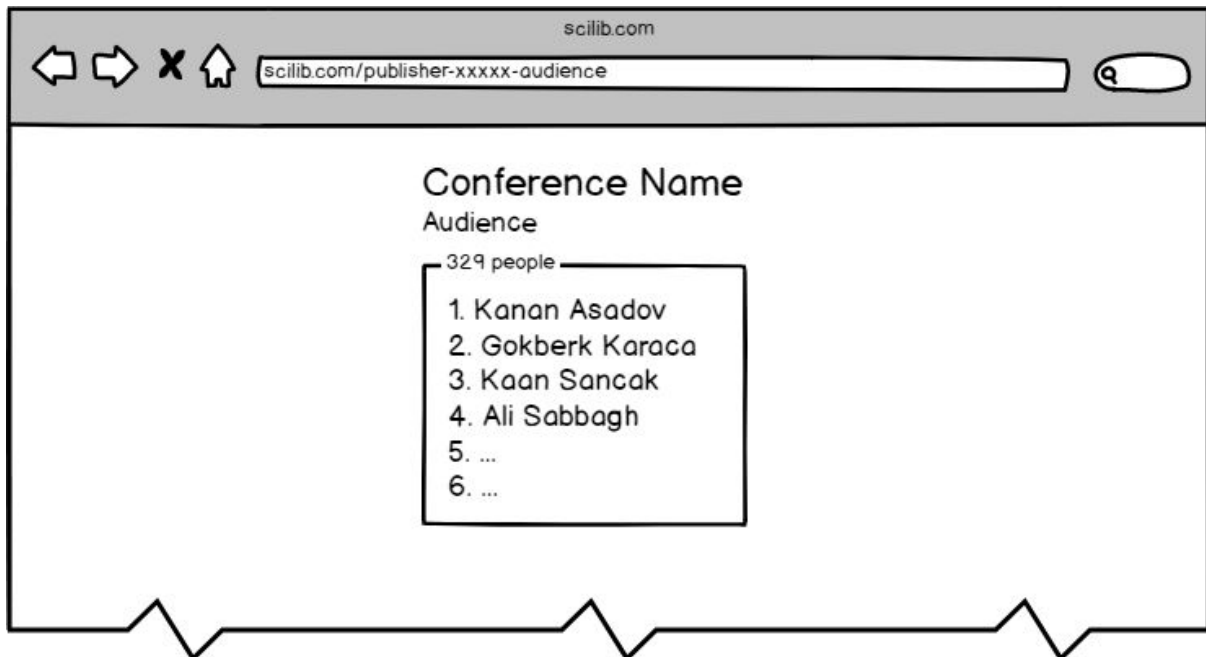
**Getting a submission reviewers:**

```
SELECT email, concat(s_name , " " , s_surname) as Name,  
num_total_reviews, ref_institution_name  
FROM invites, reviewer, subscriber  
WHERE subscriber.email = reviewer.ref_subscriber_email  
AND invites.ref_reviewer_email = reviewer.ref_subscriber_email  
AND ref_submission_id = @submission_id
```

**Sending for Author(s) approval:**

```
UPDATE submission  
SET status = 3  
WHERE s_id = @submission_id
```

## 5.13. Conference Audience Page



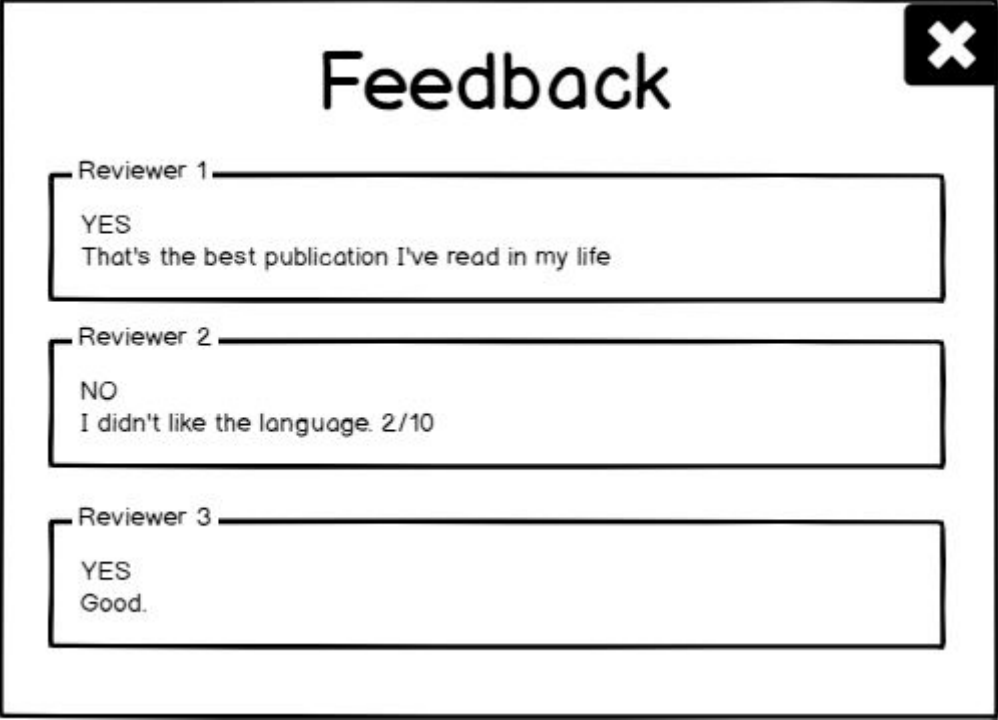
**Inputs:** @conference\_name

**Process:** To arrive to this page, the user can click on the “View Audience” button on the conference page. Then the database is queried by using the name of the conference and audience of the conference is received. Then the number of audience and their names are displayed on the conference audience view page.

**SQL Statements:**

```
SELECT a_name, a_surname  
FROM audience A  
WHERE A.ref_p_name = @conference_name
```

## 5.14. Feedback Popup



The image shows a 'Feedback' popup window with a title bar and a close button (X) in the top right corner. Inside the window, there are three separate boxes, each representing a reviewer's feedback. Each box has a label 'Reviewer 1', 'Reviewer 2', and 'Reviewer 3' respectively, followed by their comments.

Reviewer	Feedback
Reviewer 1	YES That's the best publication I've read in my life
Reviewer 2	NO I didn't like the language. 2/10
Reviewer 3	YES Good.

**Inputs:** @ref\_submission\_id

**Process:** This pop-up window can be opened either by the Editor on the Editor Submissions View by clicking the “See Feedback” button, or the Author in the Author Submissions View by clicking the “See Feedback” button. The feedbacks of the reviewers are obtained from the database and displayed here.

### SQL Statements:

#### Displaying feedbacks with names:

```
SELECT A.feedback, C.s_name
FROM reviews A, reviewer B, subscriber C
WHERE A.ref_submission_id = @ref_submission_id
AND A.ref_reviewer_email = B.ref_subscriber_email
AND B.ref_subscriber_email = C.email
```

## 6. Advanced Database Components

### 6.1. Views

#### 6.1.1. Main Page View

We wanted to limit the access of users to publications information. To do this we decided to create a view which returns only the id's of the papers, titles and the publisher names.

```
CREATE VIEW main_page_view AS
SELECT P.p_id, P.title, S.ref_publisher_name
FROM (publication P natural join submits S), author A, subscriber
B
WHERE ref_author_email = A.ref_subscriber_email AND
      A.ref_subscriber_email = B.email
```

The following 3 views have the same purpose. We decided to create them for security purposes. We needed to hide some attributes of tables from different user types. For example we did not want to let "Authors" access the submission or submits table.

#### 6.1.2. Author Submissions View

```
CREATE VIEW author_submissions_view AS
SELECT s_id, title, ref_publisher_name, status
FROM submission, submits
WHERE ref_submission_id = s_id
AND ref_author_email = @user_email
```

#### 6.1.3. Editor Submissions View

```
CREATE VIEW editor_submissions_view AS
SELECT s_id, title, getAuthorsOfSubmission( s_id ), date,
ref_publisher_name
FROM submissions, submits
WHERE ref_submission_id = s_id
AND ref_editor_email = @user_email
AND status = 0
```

### 6.1.4. Reviewer Submissions View

```
CREATE VIEW reviewer_submissions_view AS
SELECT S.title, P.name, P.topic
FROM invites I, submission S, submits S2, journal J
WHERE I.ref_reviewer_email = @reviewer_email
AND I.ref_submission_id = S.s_id AND S.s_id = S2.ref_submission_id
AND S2.ref_publisher_name = J.ref_publisher_name
```

## 6.2. Stored Procedures

In our Scientific Paper Library application, we plan to use procedures while creating new accounts for the users. On the sign up page, it is possible to create different types of user accounts. Independent from the account type, a new row is added to the subscriber table. Depending on the type of the account, different procedure calls are performed and a new row is added to the related table. The procedures for this part will be as follows;

### 6.2.1. Author Signup

```
CREATE PROCEDURE authorSignup
(@email, @password, @s_name, @s_surname,
@field_of_expertise_table)
AS
BEGIN
INSERT INTO subscriber VALUES (@email, @password, @s_name,
@s_surname, @institution);
INSERT INTO author(ref_subscriber_email) VALUE (@email);
INSERT INTO authorExpertise
FROM @email AS ref_subscriber_email, @field_of_expertise_table AS
tag;
END
```

### 6.2.2. Editor Signup

```
CREATE PROCEDURE editorSignup
(@email, @password, @s_name, @s_surname,
@field_of_expertise_table)
AS
BEGIN
INSERT INTO subscriber VALUES (@email, @password, @s_name,
@s_surname, @institution);
```

```
INSERT INTO editor(ref_subscriber_email) VALUE (@email);
END
```

### 6.2.3. Reviewer Signup

```
CREATE PROCEDURE reviewerSignup
(@email, @password, @s_name, @s_surname,
@field_of_expertise_table)
AS
BEGIN
INSERT INTO subscriber VALUES (@email, @password, @s_name,
@s_surname, @institution);
INSERT INTO reviewer(ref_subscriber_email) VALUE (@email);
INSERT INTO reviewerExpertise
FROM @email AS ref_subscriber_email, @field_of_expertise_table AS
tag;
END
```

### 6.2.4. Publish Paper

```
CREATE PROCEDURE publish_paper
( @title_to_publish, @pages, @submission_id )
AS
BEGIN
INSERT INTO publication (title, pages, publication_date,
downloads, ref_submission_id )
values(@title_to_publish, @pages, CURDATE(), 0, @submission_id )
UPDATE submission
SET status = 4
WHERE s_id = @submission_id
END
```

### 6.2.5. Submit Paper

submit\_paper( @new\_title, @new\_editor)

We will have a procedure which will add the co-authors of the given submission the submits relationship.

### 6.2.6. Get Authors of Submission

getAuthorsOfSubmission( s\_id)



We will also use another stored procedure to add sponsors to the publication. Since a publication can have more than one sponsors, the same process is repeated easily when stored procedures are used.

## 6.3. Reports

### 6.3.1. Average number of authors per publication for every publisher

```
with num_authors_publication
( publication_id, num_of_authors )
AS (
    Select ref_submission_id, count( ref_author_email )
    From submission, publication, submits
    WHERE s_id = publication.ref_submission_id
    AND s_id = submits.ref_submission_id
    GROUP BY ref_submission_id
)

Select ref_publisher_name, avg( num_of_authors )
From num_authors_publication, submits
WHERE publication_id = ref_submission_id
GROUP BY ref_publisher_name
```

## 6.4. Triggers

- When a subscriber downloads a publication, a trigger will increase the publication download count by 1.
- When a reviewer submits feedback, a trigger checks if all other reviewers assigned to this submission have submitted their feedback, if yes changes the submission status to 2, to allow editor to have a last check.
- When a new publication is added, a trigger increases num\_of\_publications for every author and his avg\_citations and num\_of\_citations for every cited paper's author.
- When an editor approves feedbacks from reviewers, a trigger will increase the num\_total\_reviews for every reviewer.
- On the first day of every year, a trigger increases editors' experience.
- When a submission is canceled, a trigger deletes the submission related entries from the invites, review and submits tables.

## 6.5. Constraints

- A user cannot login without creating a account with a unique e-mail address.
- A subscriber cannot login its account if the entered password is not matched.
- An e-mail address cannot be used for more than one subscriber account.
- An author cannot cancel his submission after it has been published.
- An author cannot make a submission to a journal/conference which is not related to author's field of interests.
- A publication should be published in only one journal or conference.
- A submission should be approved by both editors and reviewers to be published.
- An editor can only assign a reviewer to a submission who has an expertise field included in the submission fields.
- A reviewer cannot be exits without a field of expertise
- An existing audience must have a conference to attend or should have attended a conference before.
- An author must have at least one field of expertise.
- A publisher without any editors is not possible.
- A editor without a publisher is not possible..

## 7. Implementation Plan

We chose MySQL as the database management system of our application because it is free to use, easy to learn and well supported. We will use PHP and JavaScript as the implementation languages of user interface and application logic.