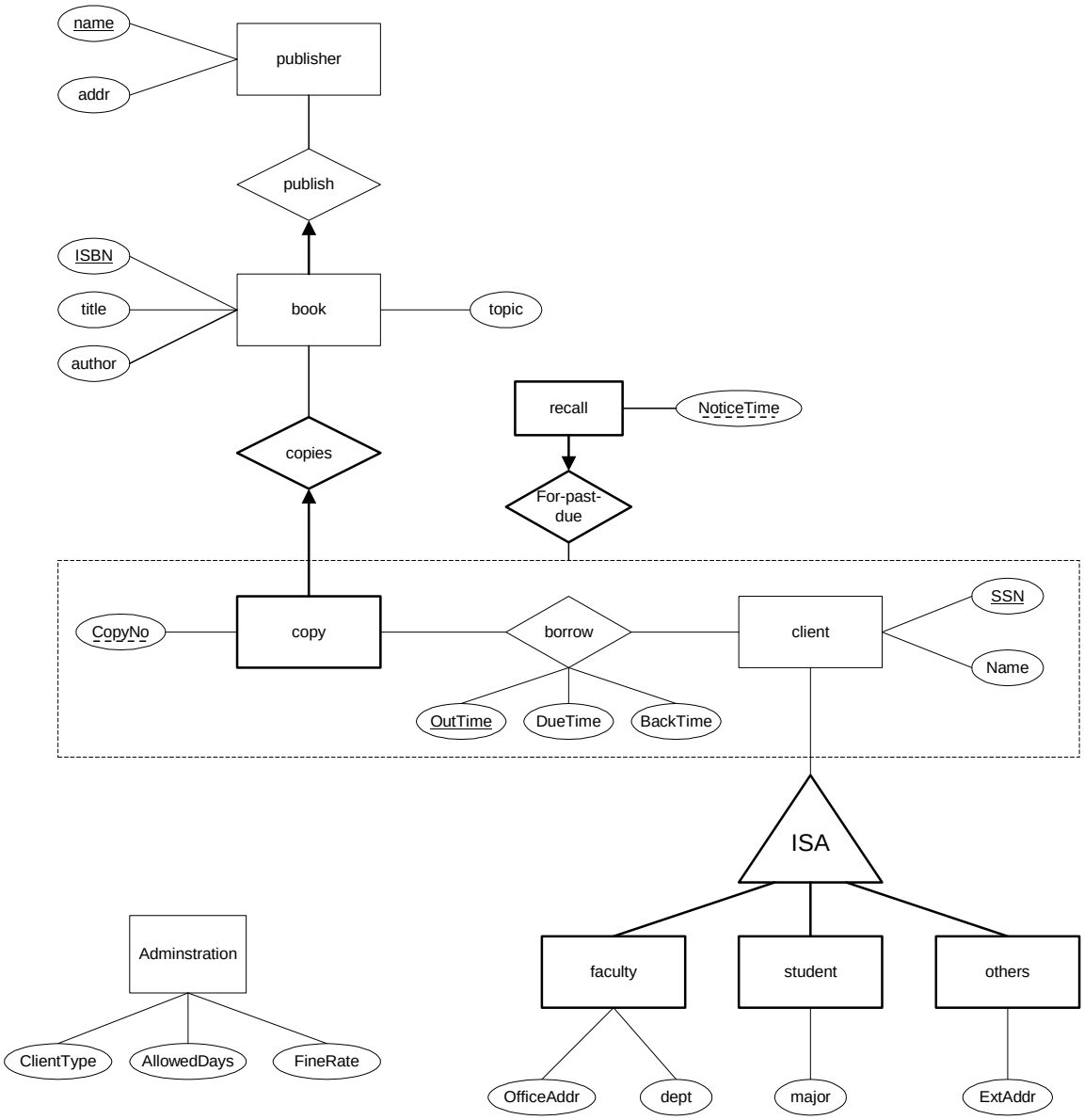
# E-R diagram for question 1:



# DDL for question 1:

```
CREATE TABLE  PUBLISHER (
        NAME        CHAR(60),
        ADDR        CHAR(60),
        PRIMARY KEY (NAME)  )

/* We assume a book has only one author. It's reasonable since no multiple authors for
one book is mentioned in the given queries */
CREATE TABLE  BOOK (
        ISBN        CHAR(10),
        TITLE       CHAR(80),
        AUTHOR      CHAR(40),
        TOPIC       CHAR(20),
        PUBLISHER   CHAR(60) NOT NULL,
        PRIMARY KEY (ISBN),
        FOREIGN KEY (PUBLISHER) REFERENCES   PUBLISHER(NAME) )

CREATE TABLE COPY (
        ISBN        CHAR(10),
        COPYNO      INTEGER,
        PRIMARY KEY (ISBN, COPYNO),
        FOREIGN KEY (ISBN) REFERENCES  BOOK ON DELETE CASCADE )

CREATE TABLE CLIENT (
        SSN         CHAR(9),
        NAME        CHAR(40),
        PRIMARY KEY (SSN)  )

CREATE TABLE  FACULTY (
        SSN         CHAR(9),
        OFFICE      CHAR(60),
        DEPT        CHAR(20),
        PRIMARY KEY (SSN),
        FOREIGN KEY (SSN) REFERENCES  CLIENT  ON DELETE CASCADE )

CREATE TABLE  STUDENT (
        SSN         CHAR(9),
        MAJOR       CHAR(20),
        PRIMARY KEY (SSN),
        FOREIGN KEY (SSN) REFERENCES  CLIENT  ON DELETE CASCADE )

CREATE TABLE  OTHERS (
        SSN         CHAR(9),
```

```
        EXT_ADDR   CHAR(60),
        PRIMARY KEY (SSN),
        FOREIGN KEY (SSN) REFERENCES  CLIENT  ON DELETE CASCADE )

CREATE TABLE  BORROW (
        SSN             CHAR(9),
        ISBN            CHAR(10),
        COPYNO          INTEGER,
        OUTTIME         DATE,
        DUETIME         DATE,
        BACKTIME        DATE,
        PRIMARY KEY (SSN, ISBN, COPYNO, OUTTIME),
        FOREIGN KEY (SSN) REFERENCES  CLIENT,
        FOREIGN KEY (ISBN, COPYNO) REFERENCES  BOOK )

CREATE TABLE  RECALL (
        SSN                     CHAR(9),
        ISBN                    CHAR(10),
        COPYNO                  INTEGER,
        OUTTIME                 DATE,
        NOTICETIME              DATE,
        PRIMARY KEY (SSN, ISBN, COPYNO, OUTTIME, NOTICETIME),
        FOREIGN KEY (SSN, ISBN, COPYNO, OUTTIME) REFERENCES
BORROW )  )

/*
The following table is for deciding clients' allowed number of days of borrowing books,
and fine rate based on their client type. Whenever we need to calculate allowed period of
borrowing book or amount of fine for a specific type of person, we need to join this table
with other related tables. If we store such information in each person, too much space
will be wasted. And update on such information will be more difficult.
*/
CREATE TABLE  ADMINSTRATION (
        CLIENT_TYPE     CHAR(1),      /* 'F'=Faculty, 'S' =Student, 'O'=Others */
        ALLOWED_DAYS    INTEGER,
        FINE_RATE       NUMERIC(5, 2) )
```

# Implementation of required queries:

Most of the given queries/constraints can be easily implemented in SQL. However, some of them have some difficulties or alternative choices in SQL:

- What is the address of the publisher of Gone with the Wind?"

```
        select      p.addr
```

```
from    publisher p, book b
where   b.publisher = p.name and b.title=' Gone with the Wind'
```

- Provide the names of people who are currently borrowing any book written by Stephen King."

```
select    c.name
from      client c, borrow br, book b
where     c.ssn = br.ssn and br.isbn=b.isbn
          and br.backtime=null and b.author=' Stephen King'
```

- How many copies of Origin of Species are currently out on loan?"

```
Select    count(br.copyno)
From      borrow br, book b
Where     br.isbn=b.isbn  and br.backtime=null
  and     b.author=' Origin of Species '
```

- *How much does Professor Deadwood currently owe in fines?" Our library has three categories of clients: faculty, student and others. They differ in how they are contacted (e.g. office, external address) Also, the period for which they are allowed to take out a book and the rate at which they are fined for overdue books.*

With SQL DDL, We can't easily represent the disjoint constraint among entity Faculty, Student, Others, which means each client must belong to at most one of Faculty, Student, or Others. Also we can represent the covering constraint of entity Client on Faculty, Student, Others, which means each client must belong to at least one of table Faculty, Student, or Others. The only way we can implement it in SQL is to use **assertion** as follows:

```
/* For disjoint constraint */
CREATE ASSERTION  F_S_O_DISJOINT
CHECK (    (  SELECT COUNT(*)
              FROM  Faculty F, Student S, Others O
              WHERE  F.SSN=S.SSN  OR F.SSN=O.SSN  OR S.SSN=O.SSN )  =  0   )

/* For covering constraint */
CREATE ASSERTION  C_COVER_F_S_O
CHECK (    (  SELECT COUNT(*)
              FROM  Client C
              WHERE
                    NOT EXISTS ( SELECT *
                                 FROM Faculty F
                                 WHERE F.SSN=C.SSN)
              AND   NOT EXISTS ( SELECT *
                                 FROM Student S
                                 WHERE S.SSN=C.SSN)
              AND   NOT EXISTS ( SELECT *
```

```
                    FROM Others O
                    WHERE O.SSN=C.SSN)  )  =  0  )
```

Although it's doable, if we create the above assertion, each insert/update/delete on those tables will cause constraint checking, thus the database performance will be significantly affected. Unless really needed, such assertion is not recommended.

- When did we issue a recall notice for Principia Mathematica from Lisa Simpson, and has it been returned yet?"

The reason the recall and borrow in so complicated in E-R and DDL is that I assume we need to keep a recall history. If we don't care about the previous recalls for an unreturned book, i.e, we just need the more recent recall, then design can be much simplified.

In E-R diagram, we can replace the weak entity (with relationship "for-past-due") with adding a new attribute "recall_time" in entity borrow. Similarly in SQL DDL, we replace the creation of table BORROW and RECALL with the following:

```
CREATE TABLE  BORROW (
        SSN             CHAR(9),
        ISBN            CHAR(10),
        COPYNO          INTEGER,
        OUTTIME         DATE,
        DUETIME         DATE,
        BACKTIME        DATE,
        RECALLTIME      DATE,
        PRIMARY KEY (SSN, ISBN, COPYNO, OUTTIME),
        FOREIGN KEY (SSN) REFERENCES  CLIENT,
        FOREIGN KEY (ISBN, COPYNO) REFERENCES  BOOK )
```

- For each major offered in the university, provide a numerical estimate of the average amount of use of the library made by students in that major." It is up to you to define the numerical estimate, but it should be based on the history of how students have borrowed books.

```
        Select      student.major, count(*)
        From        borrow br, student s
        Where       br.ssn=s.ssn
        Group by student.major
```

- *List the titles of books we carry whose topics cover both economics.llama-breeding and computers.databases.relational." Topics are hierarchically organized and books may have more than one topic. However only "incomparable" topics are listed. Example: one would not list a book under both computers.databases.relational and computers.databases; the latter would be inferred from the former.*

1) Solution 1:
   One way to solve this problem is to store the whole inference string in table TOPIC and COVER instead of just topic, for example, we store "computers.databases.relational" instead of "relational". Then when we search by topic, we use pattern matching.

   For example, if we want to search all books on computers.databases, we can use the following query:

```
SELECT      B.TITLE
FROM        BOOK B, COVER C
WHERE       B.ISBN = C.ISBN
        AND   C.TOPIC LIKE 'computers.databases.%'
```

   We actually represent data relationship and constraints by using semantics of data. The disadvantage is obviously, we have to limit the max length of inference string when we create table. The good thing is that we don't need table INFER now, the implementation is simple, and we don't need worry about recursion.

Which solution is better depends on the actual situation.

2) Solution2:
   We build a table about the topics and table about the infer relations between topics and a table which build the relation between the books and topics.
   Ideally, we just need to store the direct inference relation between topics, for example, for computers.databases.relational, we'd like to store "computers", "databases", and "relational" in table TOPIC, and direct inference relation such as ("computers", "databases"), ("databases", "relational") in table INFER. However, if we want to search for all topics that can infer (directly or indirectly), we'll have problem because SQL doesn't support recursion. So we have to find another way to solve this problem.
   Store all "inferable" relations, which means either direct or indirect. In the above example, we also need to store inference ("computers", "relational") in table INFER. We also add a new column called IF_DIRECT to distinguish if the inference is direct or not. So we need much more space to store such derived relations. Another problem is if we need add a new topic, for example, if we want to add topic "SQL" under relational , i.e. computers.databases.relational.SQL, now besides add ("relational", "SQL") to INFER table, we also need to find all ancestors of "relational" (i.e. "computers", "databases") and add pairs between them and "SQL" into the table INFER. Thus data modification will be more complicated. If such modifications occur often, the overhead is noticeable.

```
CREATE TABLE TOPIC (
        TNAME       CHAR(20),
        PRIMARY KEY (TNAME) )
```

```
CREATE TABLE TOPIC_COVER (
        ISBN            CHAR(10),
        TOPIC           CHAR(20),
        PRIMARY KEY (ISBN, TOPIC),
        FOREIGN KEY (ISBN) REFERENCES  BOOK,
        FOREIGN KEY (TOPIC) REFERENCES  TOPIC(TNAME)  )


    CREATE TABLE TOPIC_INFER (
        FROM            CHAR(20),
        TO              CHAR(20),
        IF_DIRECT    BOOLEAN,
        PRIMARY KEY (FROM, TO),
        FOREIGN KEY (FROM) REFERENCES  TOPIC(TNAME),
        FOREIGN KEY (TO) REFERENCES  TOPIC(TNAME)  )
```

- How many times has copy 3 of Swan's Way been borrowed?"

```
        Select      count(*)
        From        borrow br, book b
        Where       br.isbn=b.isbn
        and         br.copyno= 3  and  b.title=' Swan's Way'
```

- Has a faculty member in Electrical Engineering ever borrowed a book by Voltaire?"

```
        Select      *
        From        faculty f, book b, borrow br
        Where f.ssn=br.ssn and b.isbn = br.isbn
          and       f.dept=" Electrical Engineering' and b.author='Voltaire'
```

# E-R diagram for question 2: