

CSC 540 DATABASE MANAGEMENT CONCEPTS AND SYSTEMS

WOLFCITY PUBLICATION HOUSE

Project Report 3

Team Members (Q):

Alisha Shahane (asshahan)

Shruti Kangle (sskangle)

Poorva Kulkarni (pnkulkar)

Assumptions:

a. Publications:

- i. Publications can either be books or periodic publications.
- ii. Books have chapters and periodic publications have articles.
- iii. Each publication is identified by its isbn.
- iv. Periodic publications have a type which indicate whether it is a magazine or a journal.
- v. All editions of a book or issues of a periodic publication have the same isbn.
- vi. Price is associated with every edition or issue.

b. Employees:

- i. Every employee of the publishing house is either an admin, author, editor or a journalist.
- ii. Authors have access to all chapters of the book edition he/she has contributed to.
- iii. Similarly, journalists have access to all articles of the magazine/journal issue he/she has contributed to.
- iv. Editors are allowed to edit chapters and articles only of books or journals/magazines assigned to them.
- v. The pay_date denotes the last payment date of every employee and salary indicates the last paid amount to that employee.
- vi. Type indicates whether the author, editor or journalist is a staff or invited member. Type for admins of the publishing house is 'admin'.

c. Order:

- i. A single order can contain only one issue/edition of a book/magazine. Order amount is calculated as (number of copies*price per copy).
- ii. The shipping cost is calculated as 10% of the order amount.
- iii. The distributor bears both the order amount and shipping cost for a placed order.
- iv. We maintain the status of every order, and it can be
 1. accepted
 2. completed paid
 3. completed not paid
 4. discarded
- v. The admin of the publishing house is allowed to process the orders.
- vi. When the distributor places an order, the order is placed with status "accepted" and the order amount is added to his balance due.
- vii. If the admin processes the order before the expected delivery date, then the status of the order is changed to "completed not paid".

- viii. The distributor can pay for an order only after it is completed, and he can only pay the full amount.
- ix. Once the distributor pays for the order, the status is changed to “completed paid”.
- x. If the admin processes the order after the expected delivery date, then the status of the order is changed to “discarded”. In this case the distributor does not have to pay, and his balance is updated accordingly.
- xi. The publishing house is always in a position to fulfil the order, i.e always has the number of copies which the distributor requires.

d. Transactions:

- i. There are 2 kinds of transactions recorded namely, salary payments and payments from distributors whose descriptions are “Salary Paid” and “Payment Received for <order_id>” respectively.

e. Miscellaneous:

- i. The system’s scope is limited to a single publication house.
- ii. Any add/update/delete operation will take emp_id (and sometimes emp_type) as parameters and will be performed only if that is a permitted operation for that employee type.
- iii. No transaction can be deleted or updated. A transaction can only be inserted.
- iv. While generating reports, the total revenue takes only the income from Payment of Distributors into consideration.
- v. While generating reports, the total expense takes only the salary paid to the employees into consideration.

Report 1 Corrections:

Realistic Situations using Task and Operations: (Original report page 5)

	Old	New
Situation 1:	In order to publish the next edition of a book written by an invited author, the "Editing and Publishing" task adds this book in the system, assigns the editor to work on it and after editing, the editor notifies the admin to send it for publishing and update the inventory.	A distributor "D1" wants to place an order for a particular book edition. Through the "Input orders from distributors for a book edition" operation of the system, we input the title of the book, the isbn, edition, number of copies, and expected delivery date from the distributor. This operation places an order against the distributor id for D1.
Situation 2:	A distributor wants to place an order with WolfCity publishing house. The "Distribution" task adds the distributor details to the database. The admin discusses the requirements with the distributor and places an order of 1000 copies of an edition of a book to be delivered in 3 weeks.	An author "A1" wants to add a chapter to the book written by him. With the "Edit table of contents of a publication" operation, the author can add a chapter for a book with a given isbn and edition. He must provide the details like title, topic, text, chapter_id for the newly added chapter.












Application Program Interfaces: (Original report pages 6-10)

: Old APIs

: Newly Added APIs

(In order to increase readability, we have highlighted only the headings or API names)

Editing and Publishing:

Operation	Input/Parameters	Output
 addLiterature()	ISBN, edition, emp_id, title, lit_type, periodicity, text	Confirmation
 addPublication()	isbn, title, genre, emp_id	Confirmation
 updateLiterature()	ISBN, edition, emp_id, title, lit_type, periodicity, text	Confirmation
 updatePublication()	isbn, title, genre, emp_id	Confirmation
 assignWriterToLiterature()	ISBN, edition, emp_id	Confirmation
 assignBookToAuthor()	isbn, edition, emp_id(assignee), emp_id(as signer)	Confirmation
 assignPerPubToJournalist()	isbn, issue_wm, issue_year, emp_id(assignee), emp_id(as signer)	Confirmation
 assignArticleToEditor()	isbn, issue_wm, issue_year, article_id, emp_id(assignee), emp_id(as signer)	Confirmation
 assignChapterToEditor()	isbn, edition, chapter_id, emp_id(assignee), emp_id(as signer)	Confirmation
 unassignWriterFromLiterature()	ISBN, edition, emp_id	Confirmation
 unassignBookFromAuthor()	isbn, edition, emp_id(assignee)	Confirmation

	e),emp_id(assigner)	
unassignPerPubFromJournalist()	isbn, issue_wm,issue_year, emp_id(assignee),emp_id(as signer)	Confirmation
unassignArticleFromEditor()	isbn, issue_wm,issue_year, article_id, emp_id(assignee),emp_id(as signer)	Confirmation
unassignChapterFromEditor()	isbn, edition, chapter_id, emp_id(assignee),emp_id(as signer)	Confirmation
viewAssignedLiterature()	emp_id	ISBN, edition, title, genre, lit_type, periodicity, text of all literatures assigned to editor
viewAssignedEditorialWork()	emp_id	List of articles of publications assigned to the editor
addArticle()	isbn, issue_wm, issue_year, article_id, article_date, title, topic, text, emp_id	Confirmation
deleteArticle()	isbn, issue_wm, issue_year, article_id, emp_id	Confirmation
addChapter()	isbn, edition, chapter_id, chapter_date, title, topic, text, emp_id	Confirmation
deleteChapter()	isbn, edition, chapter_id, emp_id	Confirmation

Production of a book edition or of an issue of a publication:

Operation	Input/Parameters	Output
addToInventory()	emp_id, ISBN, edition, issue_date, available_copies, booked_copies, price	Confirmation
addBook()	isbn, edition, edition_date, price, emp_id	Confirmation
addPeriodicPublication()	isbn, issue_wm, issue_year, periodicity, pub_type, issue_date, price, emp_id	Confirmation
updateInventory()	emp_id, ISBN, edition, issue_date, available_copies, booked_copies, price	Confirmation
updateBook()	isbn, edition, edition_date, price, emp_id, emp_type	Confirmation
updatePeriodicPublication()	isbn, issue_wm, issue_year, periodicity, pub_type, issue_date, price, emp_id, emp_type	Confirmation
deleteFromInventory()	emp_id, ISBN, edition	Confirmation
deleteBook()	isbn, edition, emp_id	Confirmation
deletePeriodicPublication()	isbn, issue_wm, issue_year, emp_id	Confirmation
addChapter()	isbn, edition, chapter_id, chapter_date, title, topic, text, emp_id	Confirmation
addArticle()	isbn, issue_wm, issue_year, article_id, article_date, title, topic, text, emp_id	Confirmation
updateChapter()	isbn, edition, chapter_id,	Confirmation

	chapter_date, title, topic, text, emp_id, emp_type	
updateArticle()	isbn, issue_wm, issue_year, article_id, article_date, title, topic, text, emp_id, emp_type	Confirmation
updateTextOfArticle()	isbn, issue_wm, issue_year, article_id, text, emp_id, emp_type	Confirmation
updateTextOfChapter()	isbn, edition, chapter_id, chapter_date, text, emp_id, emp_type	Confirmation
deleteChapter()	ISBN, edition, chapter_id,emp_id	Confirmation
deleteArticle()	ISBN, issue_wm,issue_year, article_id, emp_id	Confirmation
findBooksByAuthorName()	emp_name	Details of all books of that author
findArticlesByJournalistName()	emp_name	Details of articles assigned to journalists
findBooks()	search_key(genre/ issue_date /emp_name) search value	ISBN, Title, Edition of all literatures that match the search criteria
findChaptersByTopic()	emp_id, emp_type,topic	Information about all chapters for the given topic
findArticlesByTopic()	emp_id, emp_type,topic	Information about all articles for the given topic
findBooksByDate()	emp_id, emp_type, start_date, end_date	Information about all books written between the given dates
findArticlesByDate()	emp_id, emp_type, start_date, end_date	Information about all articles written between the given dates

addTransaction()	date, client_id, amount	Confirmation
addPayment()	emp_id, trans_day, trans_month, trans_year, description, amount	Confirmation
trackWriterPayment()	emp_id	Details about when payment was addressed to writers

Distribution:

Operation	Input/Parameters	Output
addDistributor()	name, dist_type, city, location, street_address, contact, person_of_contact, balance	dist_id or NULL if error
addDistributor()	dist_name, dist_type, location, city, street_addr, phone, person_of_contact, balance_due, emp_id	Confirmation
updateDistributor()	dist_id, name, dist_type, city, location, street_address,, contact, person_of_contact, balance	Confirmation
updateDistributor()	dist_name, dist_type, location, city, street_addr, phone, person_of_contact, balance_due, emp_id	Confirmation
deleteDistributor()	dist_id	Confirmation
deleteDistributor()	dist_id	Confirmation
createOrder()	dist_id, ISBN, edition, amount, exp_del_date, status, num_copies	order_id or NULL if error

orderBooks()	emp_id,dist_id, title, isbn, edition, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount	Confirmation
orderPerPub()	emp_id,dist_id, title, isbn, issue_wm, issue_year, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount	Confirmation
updateDistributorBalance()	dist_id, type, amount	Confirmation
paymentByDistributor()	dist_id, order_id	Confirmation

Reports:

Operation	Input/Parameters	Output
getDistributorData()	month	number and total price of copies of each publication bought per distributor for that month
getDistributorData()	-	number and total price of copies of each publication bought per distributor per month
getTotalRevenue()	-	total revenue of the publishing house
getTotalRevenue()	month, year	total revenue of the publishing house for that month

<code>getTotalExpenses()</code>	-	total expenses of the publishing house
<code>getTotalExpenses()</code>	month, year	total expenses of the publishing house for that month
<code>getCurrentDistributors()</code>	-	total number of distributors whose order status is "Processing"
<code>getCurrentDistributors()</code>	-	total number of distributors whose order status is "Accepted" or "Completed Not Paid"
<code>calcTotalRevenue()</code>	-	Calculates total revenue (since inception) per city, per distributor, and per location.
<code>calcTotalRevenuePerCity()</code>	-	Calculates total revenue (since inception) per city
<code>calcTotalRevenuePerLocation()</code>	-	Calculates total revenue (since inception) per location
<code>calcTotalRevenuePerDistributor()</code>	-	Calculates total revenue (since inception) per distributor
<code>calculateTotalPayments()</code>	-	Calculate total payments to the editors and authors, per time period and per work type
<code>calcTotalPaymentsPerWorkType()</code>	-	Calculate total payments to the editors and authors, per work type
<code>calcTotalPaymentsPerTimePeriod()</code>	-	Calculate total payments to the editors and authors, per time period

Miscellaneous:

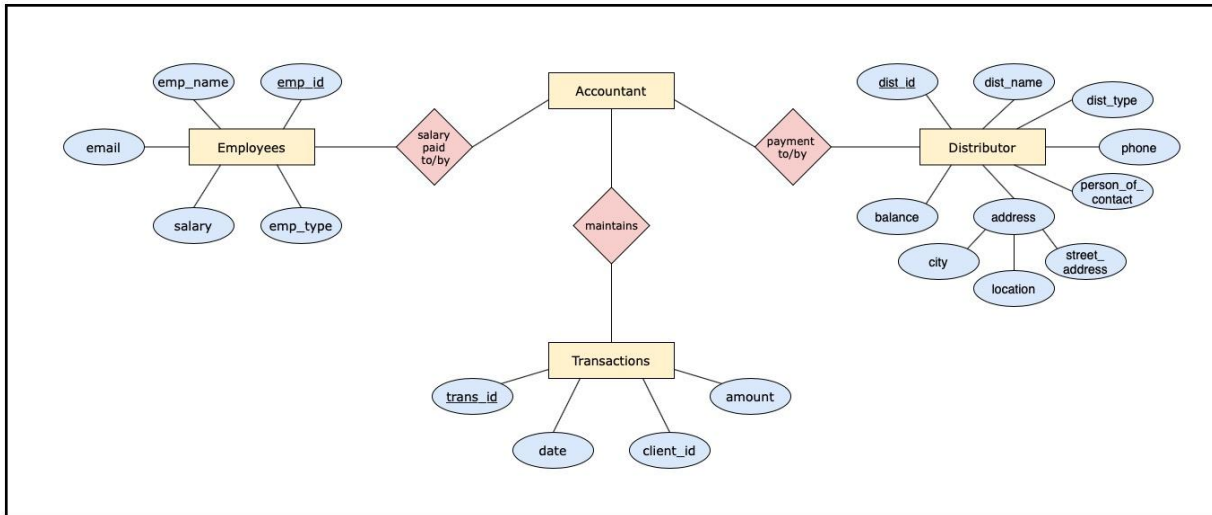
Operation	Input/Parameters	Output
processOrder()	order_id	Status updation of the order
viewMyBalance()	dist_id	Balance of the distributor
viewOrders()	emp_id/dist_id	All orders specific the admin/distributor who called the function
addWriter()	name, email, salary, emp_type, writer_type, writer_status, pay_cycle, day_of_pay	emp_id or NULL if error
addOtherStaff()	name, email, salary, emp_type	emp_id or NULL if error
addEmployee()	emp_name, email, age, gender, phone, addr, emp_type, salary	Confirmation
updateWriter()	emp_id, name, email, salary, emp_type, writer_type, writer_status, pay_cycle, day_of_pay	Confirmation
updateOtherStaff()	emp_id, name, email, salary, emp_type	Confirmation
updateEmployee()	emp_name, email, salary, emp_type, age, gender, addr, phone	Confirmation
deleteEmployee()	emp_id	Confirmation
deleteEmployee()	emp_id	Confirmation

LOCAL ER: (Original report pages 12-14)

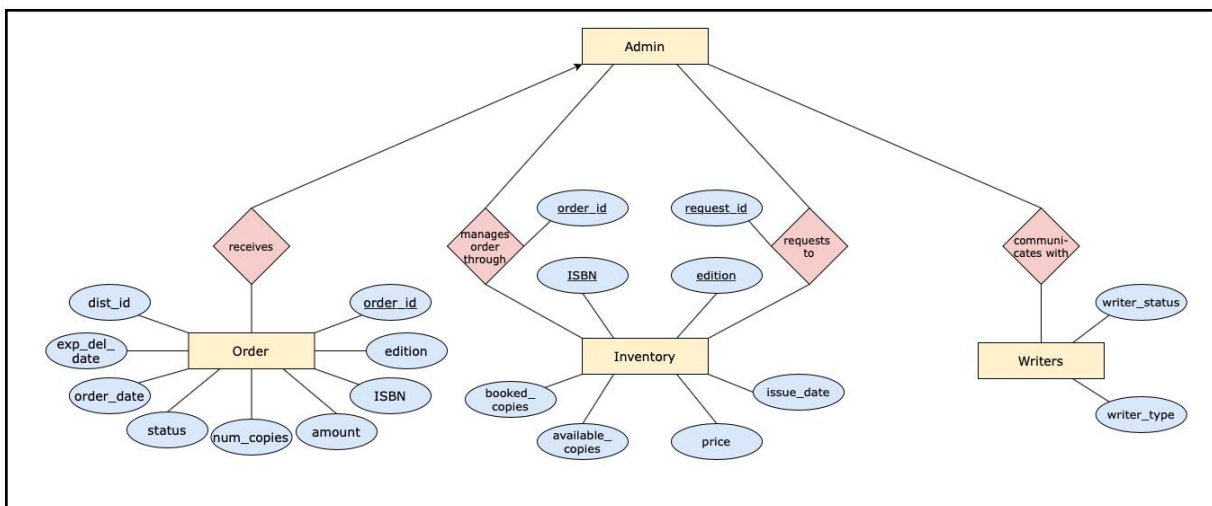
(Since we have changed the database design, we have first mentioned the old ERs and then mentioned the new ERs instead of a side-by-side layout)

Old ERs:

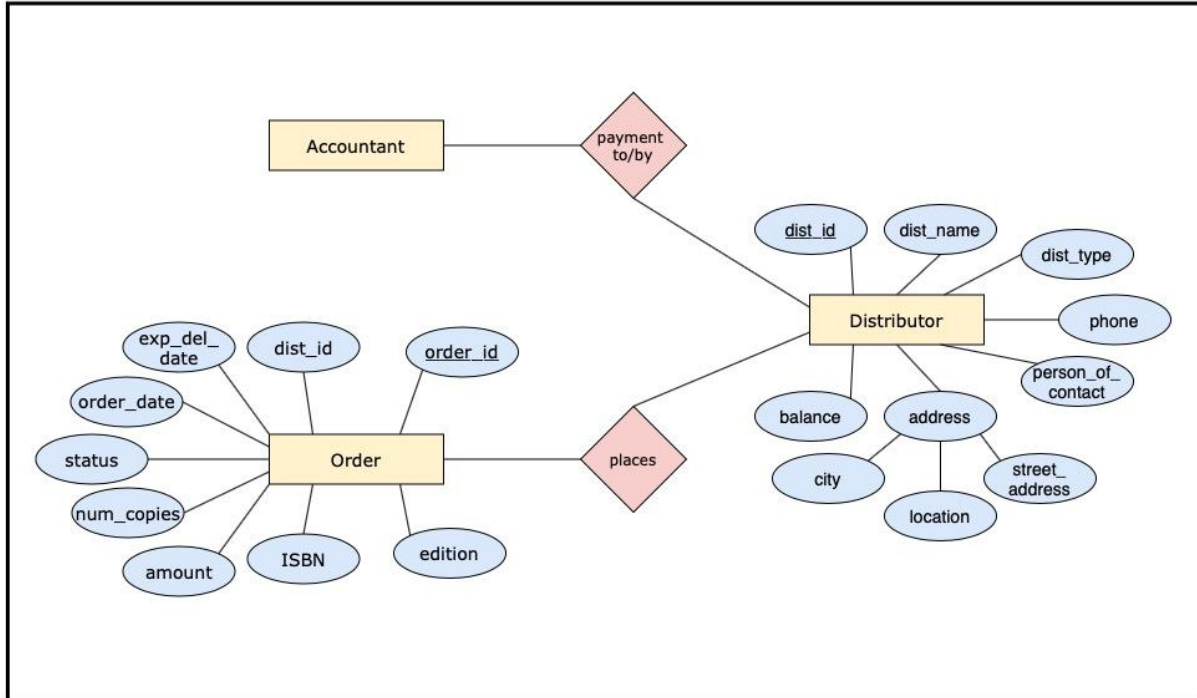
Accountant:



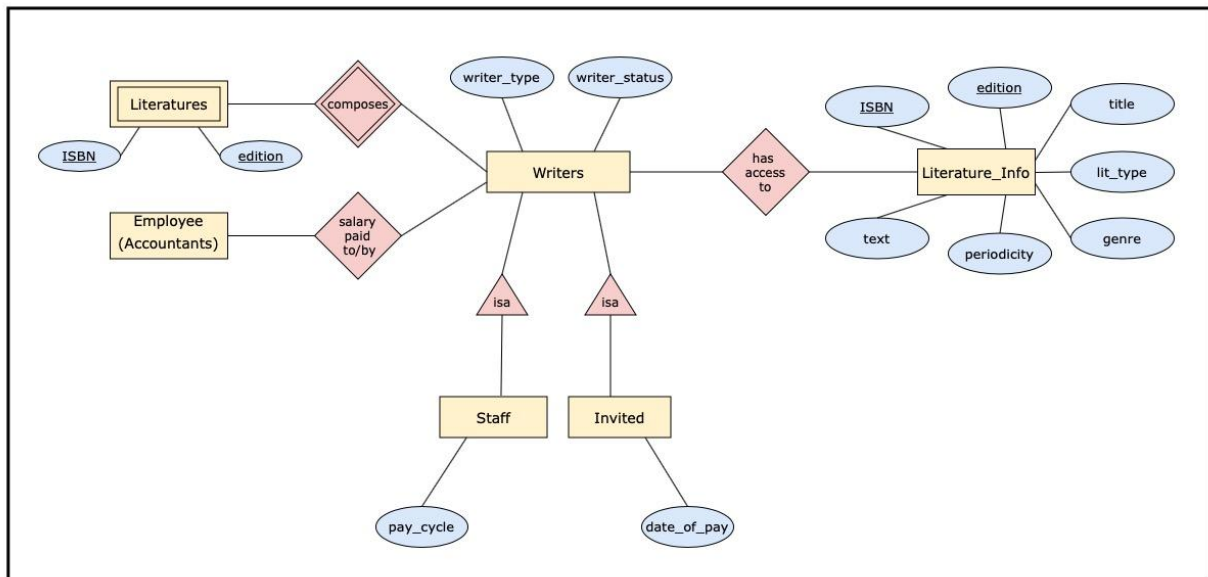
Admin:



Distributor:



Writer:



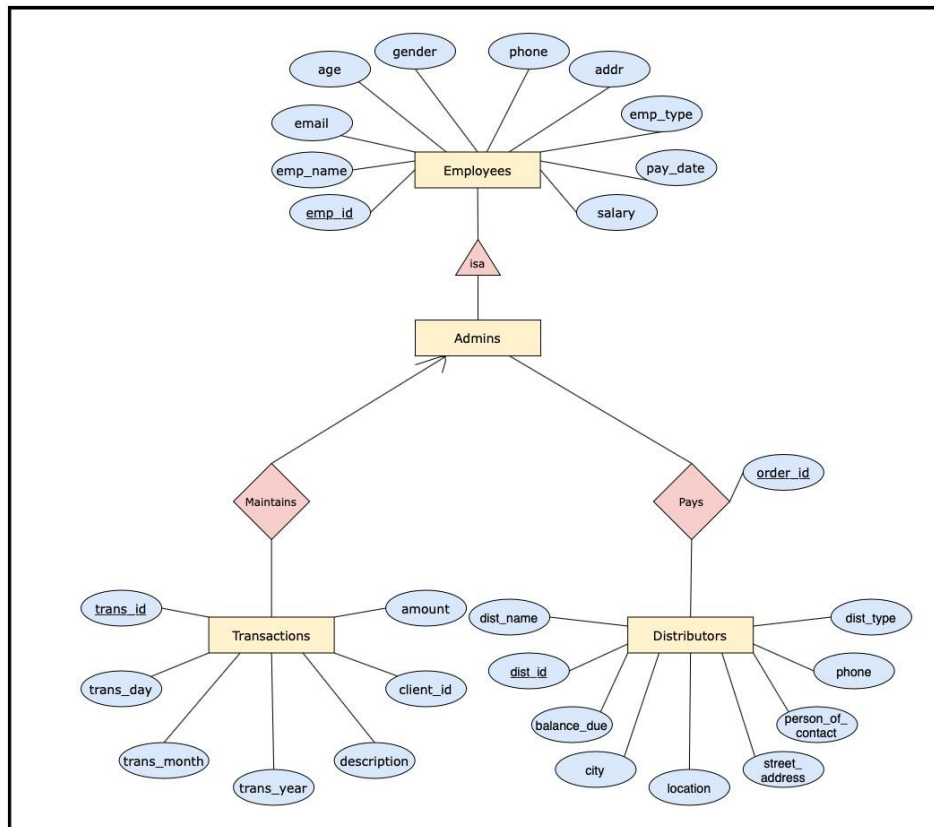
Documentation of ER Diagrams

Our database design focuses on operations of writers, accountants, admins and distributors.

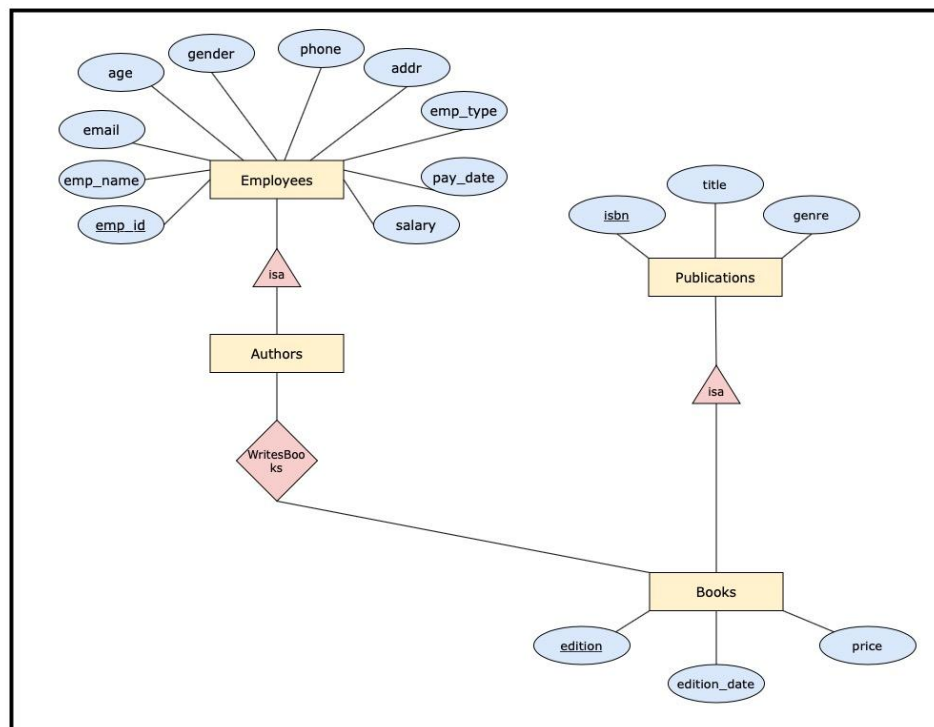
- a. Writer: A writer is a staff or an invited member.
 - i. composes: A literature is composed by an author. Literature is a weak entity as it needs to be mapped to a writer.
 - ii. has access to: A writer can access all the literature he has contributed to.
 - iii. salary paid by: Be it staff or invited, the salaries are handled by the accountant.
- b. Accountant:
 - i. salary paid to: All the employees of the publishing house are handed salary by the accountant.
 - ii. maintains: Only the accountant manages the transactions database.
 - iii. payment by: Accountant handles payments done by the distributor.
- c. Admin:
 - i. receives: Admin receives every order and maintains its status.
 - ii. communicates with: A writer notifies an admin once he/she has edited a publication.
 - iii. manages order through: Admin has access to the inventory database, and he performs activities like keeping track of available and booked copies to manage orders.
- d. Distributor:
 - i. places: The distributor places an order with the publishing house which contains their requirements of any literary work.
 - ii. payment to: The distributor settles the balance for the placed orders with the accountant of the publishing house.

New ERs:

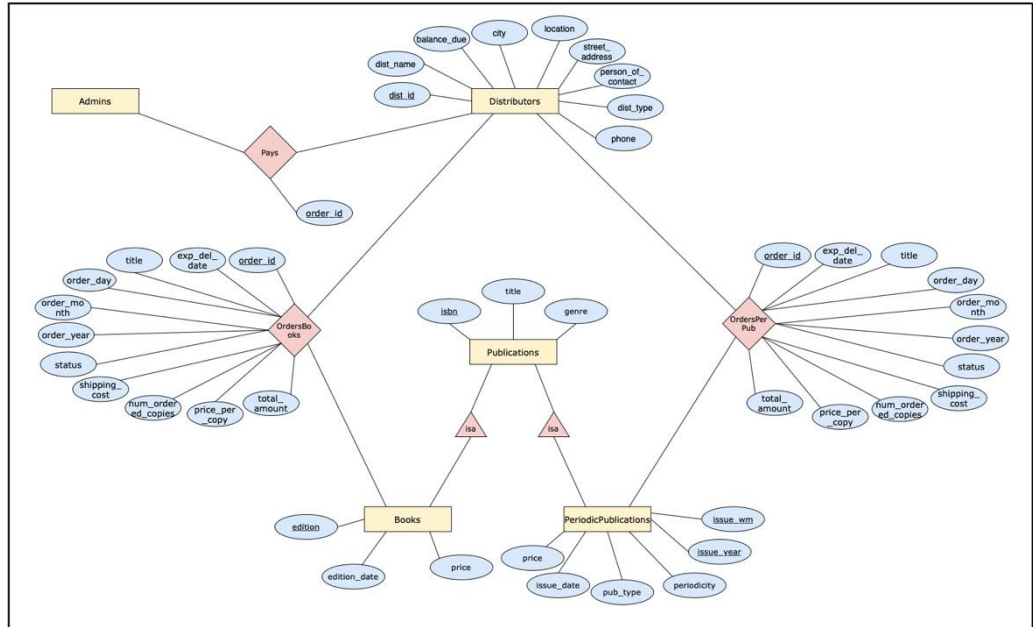
Admin:



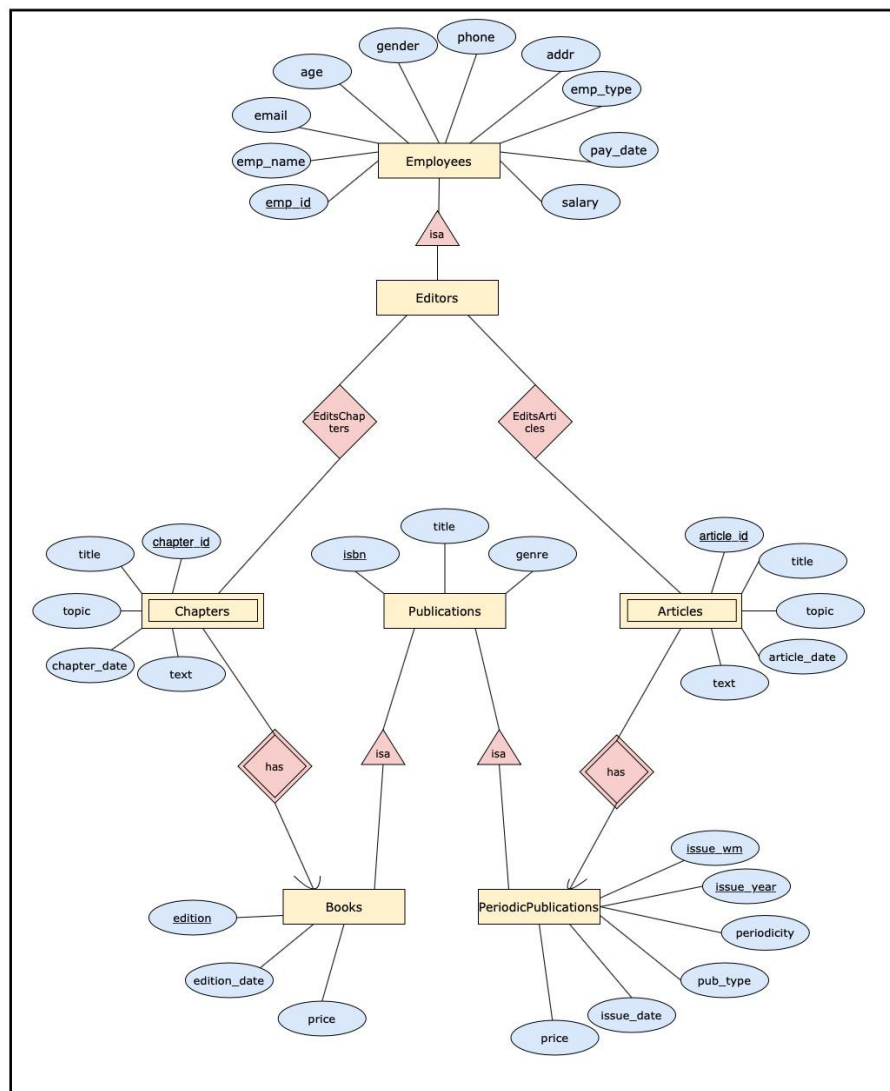
Authors:



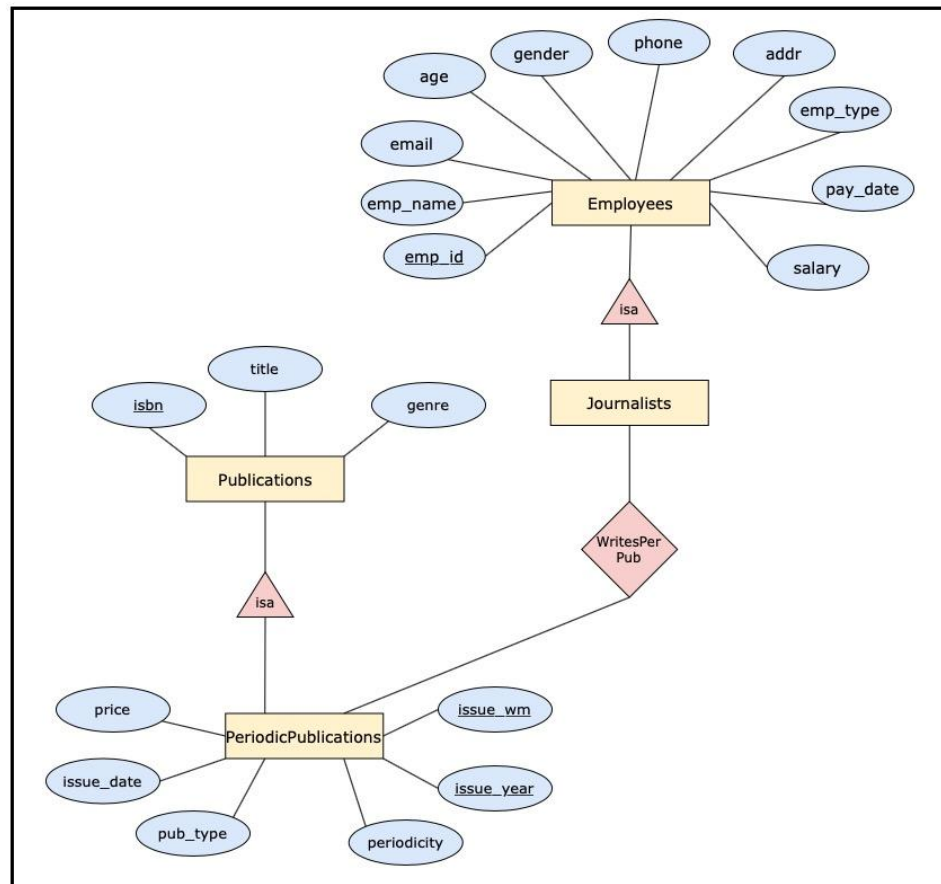
Distributors:



Editors:



Journalists:



Document Local ER

Our database design focuses on operations of editors, authors, admins, journalists and distributors.

a. Author:

- i. An author is an employee and is identified by an unique 'emp_id'.
- ii. An author may be either staff or invited which is denoted by the 'emp_type' attribute.
- iii. WritesBooks: An author writes books, which are identified by 'isbn' and 'edition'
- iv. The author has access to edit/view only those books he/she is responsible for.
- v. salary received: Be it staff or invited, the salaries are paid to the authors by the admin of the publishing house on a particular 'pay_date'.

b. Editor:

- i. An editor is an employee and is identified by a unique 'emp_id'.
- ii. An editor may be either staff or invited which is denoted by the 'emp_type' attribute.
- iii. EditsChapters, EditsArticles: The editor is responsible for editing chapters of a book or articles of a magazine/journal.
- iv. Chapters are identified by the 'chapter_id', 'edition' and 'isbn'.
- v. Articles are identified by 'article_id', 'issue_wm', 'issue_year' and 'isbn'.
- vi. The editor has access to edit/view only those or articles he/she is responsible for.
- vii. salary received: Be it staff or invited, the salaries are paid to the editors by the admin of the publishing house on a particular 'pay_date'.

c. Journalist:

- i. A journalist is an employee and is identified by an unique 'emp_id'.
- ii. A journalist may be either staff or invited which is denoted by the 'emp_type' attribute.
- iii. WritesPerPub: A journalist writes periodic publications which are of 2 types - magazines and journals which are identified by 'isbn', 'issue_wm' and 'issue_year'.
- iv. The journalist has access to edit/view only those periodic publications he/she is responsible for.
- v. salary received: Be it staff or invited, the salaries are paid to the journalists by the admin of the publishing house on a particular 'pay_date'.

d. Admin:

- i. Admin is an employee of the publishing house and is identified by a unique 'emp_id'.
- ii. Maintains: Admin maintains all the transactions incurred by the publishing house and each transaction is identified by its unique 'trans_id'.
- iii. Each transaction is maintained by exactly one admin of the publishing house.
- iv. Pays: Admin is responsible for receiving payment with respect to each order from the distributor. This payment can be characterized by a unique 'order_id'.
- v. The admin is also responsible for paying 'salary' to all the employees of the publishing house which include authors, editors, journalists and admins.

e. Distributor:

- i. The distributor orders books or periodic publications and is identified by a unique 'dist_id'. A distributor can be a wholesale distributor, bookstore or library denoted by the 'dist_type' field.
- ii. OrdersBooks: The distributor places orders of books with the publishing house which contains requirements of those books as 'num_ordered_copies'.
- iii. OrdersPerPub: The distributor places orders of books with the publishing house which contains requirements of those books as 'num_ordered_copies'.
- iv. Each order is uniquely identified by 'order_id'.
- v. Pays: The distributor settles the 'balance_due' for the placed orders with the admin of the publishing house.

Derive Local Relational Schema

1) Admin:

- ⇒ Employees (emp_id, emp_name, email, salary, pay_date, emp_type, age, gender, addr, phone)
- ⇒ Admins (emp_id)
- ⇒ Maintains(emp_id, trans_id)
- ⇒ Pays(dist_id, emp_id, order_id)
- ⇒ Distributors (dist_id, dist_name, dist_type, location, city, street_addr, phone, person_of_contact, balance_due)
- ⇒ Transactions (trans_id, trans_day, trans_month, trans_year, client_id, amount, description)

2) Authors:

- ⇒ Employees (emp_id, emp_name, email, salary, pay_date, emp_type, age, gender, addr, phone)
- ⇒ Authors (emp_id)
- ⇒ WritesBooks (emp_id, isbn, edition)
- ⇒ Books (isbn, edition, edition_date, price)
- ⇒ Publications (isbn, title, genre)

3) Editors:

- ⇒ Employees (emp_id, emp_name, email, salary, pay_date, emp_type, age, gender, addr, phone)
- ⇒ Editors (emp_id)
- ⇒ Chapters (isbn, edition, chapter_id, chapter_date, title, topic, text)
- ⇒ Articles (isbn, issue_wm, issue_year, article_id, article_date, title, topic, text)
- ⇒ EditsChapters (emp_id, isbn, edition, chapter_id)
- ⇒ EditsArticles (emp_id, isbn, issue_wm, issue_year, article_id)
- ⇒ Publications (isbn, title, genre)
- ⇒ Books (isbn, edition, edition_date, price)
- ⇒ PeriodicPublications (isbn, issue_wm, issue_year, periodicity, pub_type, issue_date, price)

4) Journalists:

- ⇒ Employees (emp_id, emp_name, email, salary, pay_date, emp_type, age, gender, addr, phone)
- ⇒ Journalists (emp_id)
- ⇒ PeriodicPublications (isbn, issue_wm, issue_year, periodicity, pub_type, issue_date, price)
- ⇒ WritesPerPub (emp_id, isbn, issue_wm, issue_year)
- ⇒ Publications (isbn, title, genre)

5) Distributors:

- ⇒ Distributors (dist_id, dist_name, dist_type, location, city, street_addr, phone, person_of_contact, balance_due)
- ⇒ OrdersBooks (order_id, isbn, edition, dist_id, title, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount)
- ⇒ OrdersPerPub (order_id, isbn, issue_wm, issue_year, dist_id, title, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount)
- ⇒ Books (isbn, edition, edition_date, price)
- ⇒ PeriodicPublications (isbn, issue_wm, issue_year, periodicity, pub_type, issue_date, price)
- ⇒ Pays(dist_id, emp_id, order_id)
- ⇒ Admins (emp_id)
- ⇒ Publications (isbn, title, genre)

Report 2 Corrections:

Database Schemas: (Original report pages 4-7)

1. Publications (isbn, title, genre)

FDs:

isbn → title, genre

The functional dependency holds because isbn uniquely determines title and genre. That is, LHS of the FD is a superkey. Thus, the relation is in 3NF.

FDs that do not hold:

title → genre

The genre is not determined by the title. Hence, the given FD does not hold true.

2. PeriodicPublications (isbn, issue_wm, issue_year, periodicity, pub_type, issue_date, price)

FDs:

isbn, issue_wm, issue_year → periodicity, pub_type, issue_date, price

The functional dependency holds because we need isbn, issue_wm, issue_year together to uniquely identify periodicity, pub_type, issue_date. That is, LHS of the FD is a superkey. Thus, the relation is in 3NF.

FDs that do not hold:

periodicity → price

pub_type → issue_date

The price is not determined by periodicity, issue_date is not determined by pub_type. In such FDs, we have attributes which are irrelevant to each other and one cannot be derived from the other, neither do they uniquely identify other attributes together. Hence, 3NF does not hold true here because LHS is not the superkey and RHS is not part of the key.

3. Articles (isbn, issue_wm, issue_year, article_id, article_date, title, topic, text)

FDs:

isbn, issue_wm, issue_year, article_id → article_date, title, topic, text

The functional dependency holds because we need article_id along with isbn, issue_wm, issue_year to uniquely identify article_date, title, topic, text. Thus the FD is in 3NF as LHS is a superkey.

FDs that do not hold:

article_date → topic

title -> text

text -> article_date

In such FDs, we have attributes which are irrelevant to each other and one cannot be derived from the other, neither do they uniquely identify other attributes together.

Hence, 3NF does not hold true here because LHS is not the superkey and RHS is not part of the key.

4. Books (isbn, edition, edition_date, price)

FDs:

isbn, edition -> edition_date, price

The dependency holds true because in order to identify price and edition_date we need isbn and edition.

Thus, the FD is in 3NF.

FDs which do not hold:

edition_date -> price

The edition_date does not determine the price. Hence, the FD does not hold. Hence, 3NF does not hold true here because LHS is not the superkey and RHS is not part of the key.

5. Transactions (trans_id, trans_day, trans_month, trans_year, client_id, amount, description)

FDs:

trans_id -> trans_day, trans_month, trans_year, client_id, amount, description

The functional dependency holds true because trans_id is sufficient to uniquely identify trans_day, trans_month, trans_year, client_id, amount and description for that given transaction. Thus the FD is in 3NF as LHS is a superkey.

FDs which do not hold:

trans_day -> client_id

trans_month -> amount

trans_year -> description

amount -> trans_month

In such FDs, we have attributes which are irrelevant to each other and one cannot be derived from the other, neither do they uniquely identify other attributes together.

Hence, 3NF does not hold true here because LHS is not the superkey and RHS is not part of the key.

6. Distributors (dist_id, dist_name, dist_type, location, city, street_addr, phone, person_of_contact, balance_due)

FDs:

dist_id -> dist_name, dist_type, location, city, street_addr, phone,
person_of_contact, balance_due

The functional dependency holds true because dist_id is the key of the above relation which uniquely determines dist_name, dist_type, location, city, street_addr, phone, person_of_contact and balance_due. Thus, the FD is in 3NF as LHS is a superkey.

FDs which do not hold:

dist_type -> balance_due

dist_name -> city

person_of_contact -> phone

phone -> balance_due

In such FDs, we have attributes which are irrelevant to each other and one cannot be derived from the other, neither do they uniquely identify other attributes together. Hence, 3NF does not hold true here because LHS is not the superkey and RHS is not part of the key.

7. OrdersBooks (order_id, isbn, edition, dist_id, title, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount)

FDs:

order_id, isbn, edition, dist_id -> order_day, order_month, order_year,
exp_del_date, status, num_ordered_copies,
shipping_cost, price_per_copy, total_amount

The functional dependency holds because we need order_id, isbn, edition, dist_id to uniquely identify title, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount. Thus the FD is in 3NF as LHS is a superkey.

FDs which do not hold:

order_day -> amount

order_year -> num_ordered_copies

num_ordered_copies -> exp_del_date

status -> order_month

In such FDs, we have attributes which are irrelevant to each other and one cannot be derived from the other, neither do they uniquely identify other attributes together.

Hence, 3NF does not hold true here because LHS is not the superkey and RHS is not part of the key.

8. OrdersPerPub (order_id, isbn, issue_wm, issue_year, dist_id, title, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount)

FDs:

order_id, isbn, issue_wm, issue_year, dist_id -> order_day, order_month,
order_year, exp_del_date, status,
num_ordered_copies, price_per_copy,
shipping_cost, total_amount

The functional dependency holds because we need order_id, isbn, issue_wm, issue_year, dist_id to uniquely identify title, order_day, order_month, order_year, exp_del_date, status, num_ordered_copies, price_per_copy, shipping_cost, total_amount. Thus the FD is in 3NF as LHS is a superkey.

FDs which do not hold:

order_day -> amount

order_month -> exp_del_date

status -> num_ordered_copies

order_year -> status

In such FDs, we have attributes which are irrelevant to each other and one cannot be derived from the other, neither do they uniquely identify other attributes together.

Hence, 3NF does not hold true here because LHS is not the superkey and RHS is not part of the key.

Report 3

Transactions

1) Payment by distributor:

Function: paymentByDistributor(int dist_id)

File: Distributor.java

Code:

```
public static void paymentByDistributor(int dist_id) {
    try {
        // Initial code includes validations for checking employee type and order_id for which the
        // payment is supposed to be made.

        System.out.println("The amount which the distributor is paying: " + total_amount);

        // Setting auto-commit to false as we don't want the transaction to commit
automatically.
        conn.setAutoCommit(false);

        // Add into pays
        int count_pays = stmt.executeUpdate("insert into Pays (dist_id, emp_id, order_id) values
        ('" + dist_id + "','" + emp_id + "','" + order_id + "');");

        // Add transaction
        int count_trans = stmt.executeUpdate("insert into Transactions (trans_day, trans_month,
        trans_year, client_id, amount, description ) values ('" + todaysDate + "','" + currentMonth
        + "','" + currentYear + "','" + dist_id + "','" + total_amount + "','" + "Payment Received for \""
        + order_id + "');");

        // Update status of the order
        int count_orders = stmt.executeUpdate("update " + order_table + " set status =
        \"completed paid\" where order_id = " + order_id + ";");

        // Update balance of distributor
        int count_dist = updateDistributorBalance(dist_id, "subtract", total_amount);

        // If the 4 SQL statements are ALL successfully executed, then we can commit the
transaction, else we rollback the transaction
        if (count_trans == 1 && count_dist == 1 && count_pays == 1 && count_orders == 1) {
```

```

        conn.commit();
        System.out.println("Payment successful.");
    } else {
        conn.rollback();
        System.out.println("Payment not successful.");
    }

    // Reset auto-commit to true so that future transactions will be committed automatically
    conn.setAutoCommit(true);

    } catch (SQLException e) {
        System.out.println("Payment not successful.");
    } catch (Exception e) {
        System.out.println("Payment not successful.");
    }
}

public static int updateDistributorBalance(int dist_id, String type, float amount) {
    try {
        Statement stmt = conn.createStatement();
        int count = 0;
        if (type.equals("add")) {
            count = stmt.executeUpdate("update Distributors set balance_due = balance_due + " + amount
                + " where dist_id = " + dist_id + ";");
        } else if (type.equals("subtract")) {
            count = stmt.executeUpdate("update Distributors set balance_due = balance_due - " +
                amount + " where dist_id = " + dist_id + ";");
        }
        if (count == 1) {
            System.out.println("Distributor balance updated successfully.");
            return 1;
        } else {
            System.out.println("Could not update balance of distributor.");
            return 0;
        }
    } catch (SQLException e) {
        System.out.println("Could not update balance of distributor.");
        e.printStackTrace();
    }
    return 0;
}

```

Explanation:

- First, we set auto-commit to false because we don't want transactions to get committed automatically.
- The first SQL statement, updates the Pays relation by adding the distributor id, employee ID and order ID of a given order.
- Next, we create a record of this payment in the Transactions table. Further, the order status is set to "completed paid" and the distributor's pending balance is updated.
- All these changes should be committed only when all 4 statements are successfully executed which is checked by the count of rows updated on each statement.
- If this check passes, we allow the transaction to commit, else it is rolled back even if one of these 4 operations fail, to prevent the database from falling into an inconsistent state.
- Finally, auto-commit is set to true again to enable automatic commit.

2) Process an Order:

Function: paymentByDistributor(int dist_id)

File: Admin.java

Code:

```
public static void processOrder() {
    try {
        // Initial code involves taking the order_id as the input for which order we have to
        // process, along with validations pertaining to the order.

        if (exp_del_date.before(calendar.getTime())) {
            // Setting auto-commit to false as we don't want the transaction to commit
            automatically.

            conn.setAutoCommit(false);

            int count = stmt.executeUpdate("update " + order_table+ " set status =
            \"discarded\" where order_id = \"\" + order_id + \"\";");

            int count_dist = d.updateDistributorBalance(dist_id, "subtract", total_amount);

            // If the 2 SQL statements are ALL successfully executed, then we can commit
            the transaction, else we rollback the transaction
            if (count == 1 && count_dist == 1) {
                conn.commit();
                System.out.println("The order is being discarded.");
            } else {
                conn.rollback();
                System.out.println("The process request was not successful.");
            }
        } else {
            int count = stmt.executeUpdate("update " + order_table+ " set status =
            \"completed not paid\" where order_id = \"\" + order_id + \"\";");
            if (count == 1) {
                System.out.println("The order is now completed.");
            }
        }
        // Reset autocommit to true so that future transactions will be committed
        automatically.
        conn.setAutoCommit(true);
    } catch (SQLException e) {
```

```

        e.printStackTrace();
    }
}

public static int updateDistributorBalance(int dist_id, String type, float amount) {
    try {
        Statement stmt = conn.createStatement();
        int count = 0;
        if (type.equals("add")) {
            count = stmt.executeUpdate("update Distributors set balance_due = balance_due + " + amount
                + " where dist_id = " + dist_id + ";");
        } else if (type.equals("subtract")) {
            count = stmt.executeUpdate("update Distributors set balance_due = balance_due - " +
                amount + " where dist_id = " + dist_id + ";");
        }
        if (count == 1) {
            System.out.println("Distributor balance updated successfully.");
            return 1;
        } else {
            System.out.println("Could not update balance of distributor.");
            return 0;
        }
    } catch (SQLException e) {
        System.out.println("Could not update balance of distributor.");
        e.printStackTrace();
    }
    return 0;
}

```

Explanation:

- First, we set auto-commit to false because we don't want transactions to get committed automatically.
- In the first SQL statement, we are checking if the expected delivery date of the order is lesser than the current date. If that is the case, we discard the order.
- Next, we update the order table pertaining to that book/periodic publication and set its status as 'discarded'.
- The distributor must also be not billed for a discarded order. Hence, we update the distributor's balance by subtracting the order's bill amount.
- All these changes should be committed only when both statements are successfully executed which is checked by the count of rows updated on each statement.

- If this check passes, we allow the transaction to commit, else it is rolled back even if one of these 2 operations fail, to prevent the database from falling into an inconsistent state.
- Finally, auto-commit is set to true again to enable automatic commit.

3) Payment to Writer:

Function: paymentByDistributor(int dist_id)

File: Admin.java

Code:

```
public static void addWriterPayment(int emp_id) {
    try {
        // Initial code involves taking the employee id of the writer who is supposed to get paid
        // and performing other validations and authorization

        System.out.println("Salary for given employee is: " + amount);
        System.out.println("Do you want to change the amount to be paid \n1.Yes\n2.No");
        int ch = Integer.parseInt(sc.nextLine());
        int count = 0;
        int count1 = 0;

        // Setting auto-commit to false as we don't want the transaction to commit
        // automatically.
        conn.setAutoCommit(false);

        if (ch == 1) {
            System.out.println("Enter new salary:");
            amount = Float.parseFloat(sc.nextLine());

            count = stmt.executeUpdate("INSERT INTO
            Transactions(trans_day,trans_month,trans_year,client_id,amount,description)
            VALUES('"+ todaysDate + "','"+ currentMonth + "','"+ currentYear + "','"+
            emp_id + "','"+ amount + "','Salary Paid')");

            count1 = stmt.executeUpdate("UPDATE Employees SET pay_date= '"+
            sdf.format(cal.getTime())+ "',salary = '"+ amount + "' WHERE emp_id= '"+
            emp_id + "'");
        } else if (ch == 2) {

            count = stmt.executeUpdate("INSERT INTO
            Transactions(trans_day,trans_month,trans_year,client_id,amount,description)
            VALUES('"+ todaysDate + "','"+ currentMonth + "','"+ currentYear + "','"+
            emp_id + "','"+ amount + "','Salary Paid')");

            count1 = stmt.executeUpdate("UPDATE Employees SET pay_date= '"+
            sdf.format(cal.getTime())+ "',salary = '"+ amount + "' WHERE emp_id= '"+
            emp_id + "'");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```

    }
    // If the 2 SQL statements are ALL successfully executed, then we can commit the
    transaction, else we rollback the transaction
    if (count == 1 && count1 == 1) {
        conn.commit();
        System.out.println("Successfully executed transaction.");
    } else {
        conn.rollback();
        System.out.println("Could not execute transaction.");
    }
    // Reset autocommit to true so that future transactions will be committed
    automatically
    conn.setAutoCommit(true);
}
catch (SQLException e) {
    System.out.println("Could not execute transaction.");
    e.printStackTrace();
}
}

```

Explanation:

- First, we set auto-commit to false because we don't want transactions to get committed automatically.
- We first create a record of this payment to the writer in the Transactions table. Further, the salary date and amount are set to the current date and salary paid (either old or updated).
- All these changes should be committed only when all 2 statements are successfully executed which is checked by the count of rows updated on each statement.
- If this check passes, we allow the transaction to commit, else it is rolled back even if one of these 2 operations fail, to prevent the database from falling into an inconsistent state.
- Finally, auto-commit is set to true again to enable automatic commit.

High Level Design Decisions:

The database is designed to capture all the necessary information relevant to the publication.

All the relations are normalized to 3NF form. At a high level, the database tables contain information relating to publications(books and periodic publications), employees(author, journalist, editor, admin), distributors, orders and financial transactions of the publishing house. Some columns of the tables are enforced with the NOT NULL, PRIMARY KEY or FOREIGN KEY constraints.

For example, the “title” for a Publication cannot be NULL.

The “isbn” and “edition” are primary keys for the Books table. The Chapters table references “isbn” and “edition” from the Books table. The foreign key constraint prohibits adding a chapter for a book which is not present in the database. Along with the FOREIGN KEY constraint, ON DELETE CASCADE deletes all chapters of the book when a book is deleted.

Similarly, such constraints have been applied to attributes of other tables wherever necessary.

The java application facilitates accessing the database and performing various tasks and operations on it. The application is modularized into different relevant classes. The class dbConnection takes the credentials and connects to the database. The other classes namely Publication, Admin and Distributor consists of functions that are relevant to that class. The APIs have been categorized into these classes for better readability and ease of updation. The MainMenu.java file is executed to run this application. It first gives a menu which allows the user to login as Admin, Creative Staff(author, journalist or editor) or a Distributor. It also has an Exit option to terminate the application. Once the user logs in as a particular type of user, he is prompted to provide his unique ID in the system i.e. employee ID/ distributor ID. According to the type of user and the ID that the user enters, he is provided with the menu of operations that he is authorized to perform on the system. In case the user enters any incorrect information or violates any constraint of the database system, he is prompted by a message. After performing every operation, the user can choose a next operation or exit the menu to be redirected again to the login menu.

Functional Roles:

Part 1:

Role	Prime	Backup
Software Engineer	Shruti	Alisha
Database Designer / Administrator	Alisha	Poorva
Application Programmer	Poorva	Shruti
Test Plan Engineer	Alisha	Shruti

Part 2:

Role	Prime	Backup
Software Engineer	Poorva	Alisha
Database Designer / Administrator	Shruti	Poorva
Application Programmer	Alisha	Shruti
Test Plan Engineer	Poorva	Alisha

Part 3:

Role	Prime	Backup
Software Engineer	Alisha	Poorva
Database Designer / Administrator	Poorva	Shruti
Application Programmer	Shruti	Poorva
Test Plan Engineer	Shruti	Alisha