HW: Technical Diagrams

Due: Friday, April 3 @ 8pm

Table of Contents

[Entity-Relationship (ER) Diagrams](#)
- [Scenario: Restaurant Order Management](#)
- [Assignment: ER Diagram](#)

[Architecture Diagrams: Campus Printing System](#)
- [Diagram 1: Deployment Tiers](#)
- [Diagram 2: Backend Internal Layers](#)
  - [Part A](#)
  - [Part B: Trace real behaviors](#)
  - [Part C](#)

You may complete this assignment individually or with a partner. Submit your work to Canvas as a PDF. If you work with a partner, both of you should submit. Include both of your names at the top of the PDF you submit.

Entity-Relationship (ER) Diagrams

Scenario: Restaurant Order Management

A local restaurant is tired of managing orders on paper and wants a small database system to track customers, orders, and payments.

When someone orders food, the restaurant creates a customer record optionally including the customer's name and email. Every time the customer places an order, the system creates a new order with the time it was created, the order type (dine-in, takeout, or delivery), and the current status of the order (open, preparing, paid, or cancelled). Each order is placed by one and only one customer, but a customer may place many orders over time.

The restaurant has a list of items it sells on its menu. Each menu item includes a name and the current listed price. An order is made up of one or more line items. Each line item records the menu item, the quantity requested, any special instructions (like "no onions"), and the current price of that menu item at the time the order was placed (since menu prices may change later).

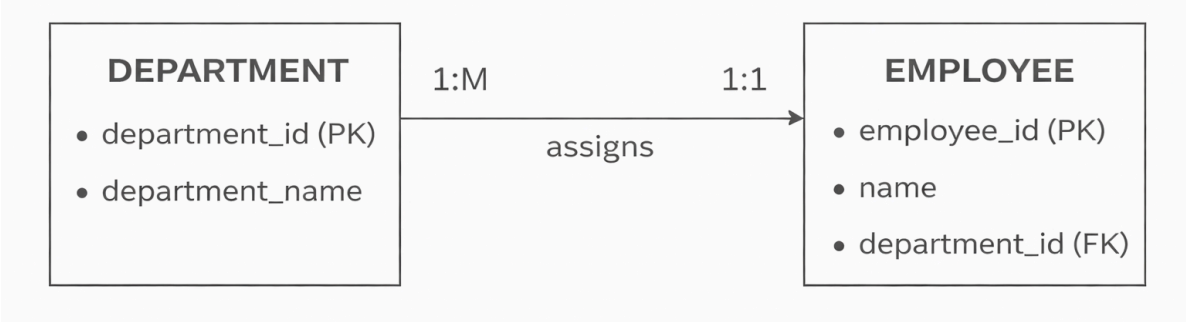
A dine-in order is assigned to one table, and takeout and delivery orders are not assigned a table. Each table has an id number and capacity, and each table may be associated with many orders over time. The restaurant may renumber tables occasionally, so the system keeps a stable internal table ID, while still storing the human-facing table number.

Finally, when the customer pays, the restaurant records a payment for the order, including the amount paid, the method (cash or card), and the time the payment occurred. An order may have zero or one payment record (an open order might not be paid yet), but every payment must correspond to exactly one order.

Assignment: ER Diagram

Make an ER diagram for the scenario described above. Your ER diagram must include the entities and attributes mentioned in the scenario, the relationships between them, and the cardinalities on those relationships. Each entity and relationship must be named. Each relationship must have two sets of cardinalities, each set containing an upper and lower bound.

Below is an example of an acceptable ER diagram format. Note that you do not need to explicitly include database-specific ID/Primary Key (PK)/Foreign Key (FK) attributes, although you may do so if you wish.



Right: Each employee must be related to exactly 1 department.
Left: Each department can be related to 0 or many employees.

“Crow’s foot” notation can also be used.

Architecture Diagrams: Campus Printing System

In this section you will analyze a simplified Campus Printing System. The university wants students to be able to submit documents to print, choose a printer, check job status (queued/printing/done/failed), and cancel jobs. The system also enforces print quotas (e.g., pages per week) and stores records of print jobs and printers.

Diagram 1: Deployment Tiers

This diagram shows the system as units of deployment (also called deployment tiers).

In this system, the three deployment tiers are:

- Client Tier (Front End): the student-facing app students use to submit and manage print jobs
- Backend Tier: a print service that enforces rules (login/auth, quotas, job creation, job status updates)
- Database Tier: persistent storage for users/quotas/printers/jobs

This view answers: “What runs where?” and “Which deployed components talk to each other?”

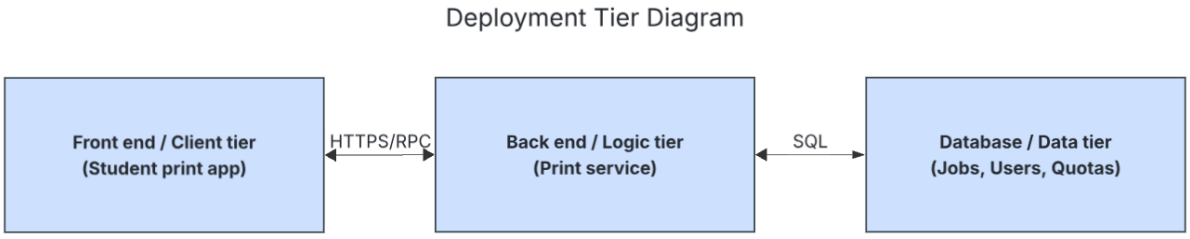


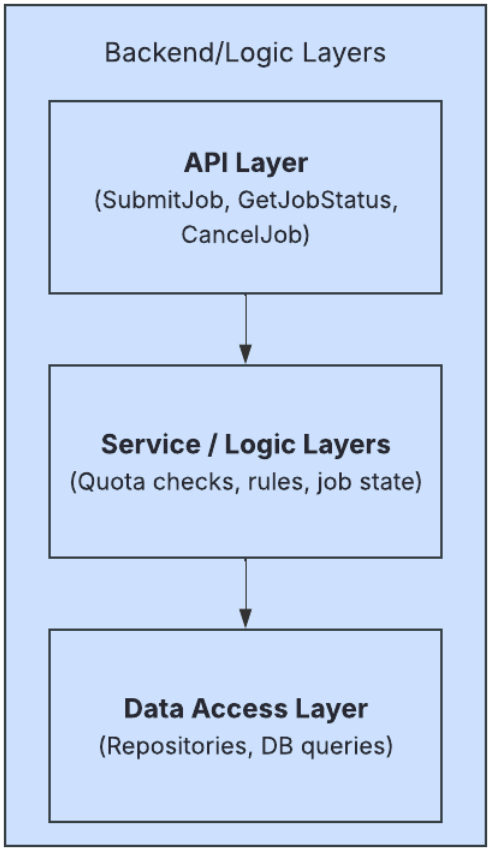
Diagram 2: Backend Internal Layers

The second diagram looks into the backend tier from Diagram 1 and shows layers inside the backend. A layer is a way to organize responsibilities within a deployed component.

For this assignment, the backend is divided into three layers:

- API Layer: receives requests (submit job, check status, cancel job) and returns responses
- Service/Logic Layer: contains the system’s rules and workflows (quota checks, permission checks, job state transitions)
- Data Access Layer: performs database reads/writes (queries, inserts, updates)

This view answers: “How is the backend organized internally?” and “Which parts of the backend depend on which?”



Part A

Explain why these statements are wrong(1-2 sentences each)

1. “The database tier is inside the backend.”
2. “The API is a separate deployment tier.”

Part B: Trace real behaviors

For each user action below, write a step-by-step trace that references boxes from the diagram.

User actions:

1. Cancel a print job
2. Check print job status

For each action include:

- A numbered list of which tier/layer is involved in each step
- What data is sent (high-level: “document”, “jobId”, “status”, “quota info”)
- Where the system of record is updated (if any)

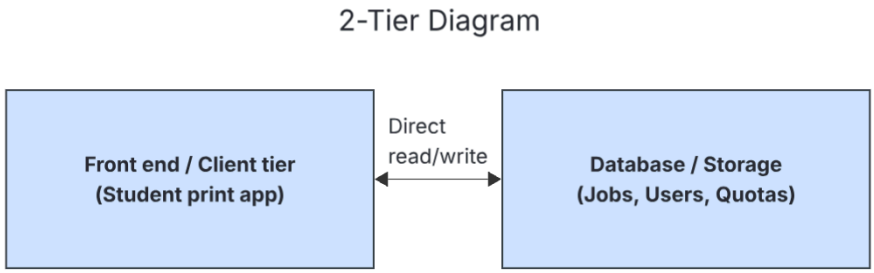
Example format:

Submit a print job

- Step 1: Client Tier → Backend API Layer: submitJob(document)
- Step 2: API Layer → Service Layer: validate request + quota check
- Step 3: Service Layer → Data Access Layer: create Job record
- Step 4: Data Access Layer → Database Tier: INSERT job …
- Step 5: Backend → Client: jobId, status=queued

Part C

Now let's assume the university instead uses printing services that read and write directly to a database.



The university adds a privacy requirement:

> Secure Release Printing: When a student submits a print job, it must be stored but NOT printed immediately. The job should print only when the student goes to a printer and authenticates at the printer (e.g., swipes a student ID card / enters a PIN) and selects which queued jobs to release.

This means a new operation needs to be added to the system:

- ReleaseJob:
    - Prints the next queued job(s) for that user at that printer
    - Updates job status to PRINTING → DONE (or FAILED)

Q1: Explain why Secure Release Printing is not supported well under the flawed architecture.

Mention at least two of these themes:

- Where the new operation would run or what additional units of deployment are needed
- Who is allowed to change job state
- Security/credentials
- Conflicting updates when two places act at once