# Arc-i-Tech Website: A Unified Database Architecture Design

## I. The Core Architectural Foundation: User and Access Management

The foundational requirement for the Arc-i-Tech platform is a database architecture capable of managing a diverse and complex set of user types. The system must accommodate corporate clients, individual students, mentors, interns, and internal Arc-i-Tech administrators, often simultaneously. The chosen design must be scalable, secure, and flexible enough to handle users who may embody multiple roles.

### A. Core Architectural Debate: Choosing the Right User Model

The design of a user management system typically follows one of three primary patterns, each with significant trade-offs.

Pattern 1: The "Single Table" Approach

This model utilizes a single, wide Users table, often with multiple boolean columns (e.g., is\_admin, is\_teacher, is\_student) to define roles.1 While this approach is simple for basic applications 2, it fails to scale. The table becomes sparse, filled with NULL values for attributes that do not apply to every role. Its most significant failure, however, is the inability to model users who hold multiple roles. For example, how would this model represent a student who is also a mentor for a different subject? The system fundamentally breaks down.

Pattern 2: The "Table-per-Subtype" (Inheritance) Approach

This model, an implementation of database inheritance, involves a base Users table for common attributes (like login credentials) and separate tables for each user type (e.g., Students, Teachers, Clients) that store role-specific columns.1 This pattern offers clean data segregation, role-specific fields, and can be optimized for performance.2 However, it shares the same critical flaw as the single-table approach: it struggles with multi-role users. A user who is both a Student and a Client would require data duplication across both tables, leading to data integrity and management crises.6

Pattern 3: The "Hybrid" or "Composition" Approach

This recommended architecture involves a single, unified Users table that stores only authentication data (email, password) and universally shared attributes (name). This table is then linked via one-to-zero-or-one relationships to separate "profile" tables (e.g., Student\_Profiles, Client\_Profiles) that store role-specific data.2 This is a "Composition over Inheritance" model.7 A User has a Student\_Profile or has a Client\_Profile. This design avoids data redundancy, is highly flexible, and is the superior pattern for this use case.2

### B. The Recommended "Composition" User Schema

The Arc-i-Tech business model implies that a single individual *must* be able to hold multiple profiles. A university student (a Student\_Profile) could also be the technical contact for a corporate team (a Client\_Profile) that is sponsoring a final-year project. An Arc-i-Tech administrator (Admin role) might enroll in a new technical course (Student\_Profile) for professional development.

This "multi-profile" reality dictates the adoption of the composition model. Research confirms the value of separating common user data (login, name) from role-specific attributes (like a student ID or a corporate job title).3 This architecture is the only one that supports Arc-i-Tech's diverse business model without forcing users to create and manage multiple accounts, an anti-pattern explicitly identified as a "nuisance to manage".12

The core Users table is thus "dissociated" from contact information and role-specific data, holding only the minimum required for a person to *exist* and *authenticate* in the system.4 The "has-a" relationships are then defined in separate profile tables, where a user\_id can be associated with any number of role-specific profiles.3

**Table Definitions:**

**Table: Users**

* **Purpose:** The single source of truth for identity and authentication.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| user\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the user. |
| email | VARCHAR(255) | NOT NULL, UNIQUE | The user's login email address. |
| password\_hash | VARCHAR(255) | NOT NULL | Hashed and salted password. |
| full\_name | VARCHAR(255) | NOT NULL | The user's full name. |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Timestamp of account creation. |
| last\_login | TIMESTAMP | NULLABLE | Timestamp of the last login. |
| is\_active | BOOLEAN | NOT NULL, DEFAULT true | Flag to enable/disable the account. |

**Table: Student\_Profiles**

* **Purpose:** Stores attributes specific to a user *as a student*.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| student\_profile\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique ID for this specific profile. |
| user\_id | BIGINT | NOT NULL, UNIQUE, FOREIGN KEY (Users.user\_id) | 1-to-1 link to the main Users table. |
| student\_identifier | VARCHAR(100) | NULLABLE, UNIQUE | Optional internal student ID number. |
| education\_level | VARCHAR(100) | NULLABLE | e.g., 'High School', 'Undergraduate'. |
| interests | JSONB | NULLABLE | Stores student's technical interests. |

**Table: Client\_Profiles**

* **Purpose:** Stores attributes specific to a user *as a corporate client contact*.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| client\_profile\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique ID for this specific profile. |
| user\_id | BIGINT | NOT NULL, UNIQUE, FOREIGN KEY (Users.user\_id) | 1-to-1 link to the main Users table. |
| client\_id | BIGINT | NOT NULL, FOREIGN KEY (Clients.client\_id) | Links to the parent organization. |
| job\_title | VARCHAR(255) | NULLABLE | The user's title at their company. |
| phone\_number | VARCHAR(50) | NULLABLE | Work-specific phone number. |

**Table: Mentor\_Profiles**

* **Purpose:** Stores attributes specific to a user *as a mentor*.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| mentor\_profile\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique ID for this specific profile. |
| user\_id | BIGINT | NOT NULL, UNIQUE, FOREIGN KEY (Users.user\_id) | 1-to-1 link to the main Users table. |
| bio | TEXT | NULLABLE | Public biography for the mentor. |
| expertise\_tags | JSONB | NULLABLE | Array of skills, e.g.,. |
| linkedin\_url | VARCHAR(255) | NULLABLE | Link to mentor's LinkedIn profile. |

### C. Role-Based Access Control (RBAC) Schema

The architecture above successfully decouples a user's *identity* (in Users) from their *data* (in ...\_Profiles). The final step is to decouple their *behavior* (their permissions).

A user's profile (e.g., Student\_Profile) defines *what they are*, while their *role* (e.g., 'Course Taker') defines *what they can do*. These are two separate concepts that must be modeled independently for a flexible system.

The gold-standard database pattern for this is Role-Based Access Control (RBAC).13 This model is built on a set of normalized tables: Users, Roles, Permissions, and two junction tables (User\_Roles, Role\_Permissions) that connect them.13 This structure allows Arc-i-Tech to define granular, atomic permissions, such as create:project\_task or view:own\_grades.16 These permissions are bundled into Roles (e.g., 'Project\_Manager', 'Student'). A User is then assigned one or more Roles via the User\_Roles junction table.4 This provides infinite flexibility, allowing a user to simultaneously have the 'Student' role (to see their courses) and the 'Client' role (to track their project).

**Table Definitions:**

**Table: Roles**

* **Purpose:** Defines the abstract roles within the system.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| role\_id | INT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the role. |
| role\_name | VARCHAR(100) | NOT NULL, UNIQUE | e.g., 'Admin', 'Client', 'Student', 'Mentor'. |

**Table: Permissions**

* **Purpose:** Defines the discrete, atomic actions a user can perform.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| permission\_id | INT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the permission. |
| permission\_name | VARCHAR(100) | NOT NULL, UNIQUE | e.g., 'project.task.create', 'course.enroll'. |

**Table: User\_Roles (Junction Table)**

* **Purpose:** Maps users to their assigned roles (Many-to-Many).
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) |  |
| role\_id | INT | NOT NULL, FOREIGN KEY (Roles.role\_id) |  |
| *Composite* |  | PRIMARY KEY (user\_id, role\_id) | Ensures a user has a role only once. |

**Table: Role\_Permissions (Junction Table)**

* **Purpose:** Maps roles to their allowed permissions (Many-to-Many).
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| role\_id | INT | NOT NULL, FOREIGN KEY (Roles.role\_id) |  |
| permission\_id | INT | NOT NULL, FOREIGN KEY (Permissions.permission\_id) |  |
| *Composite* |  | PRIMARY KEY (role\_id, permission\_id) | Ensures a permission is in a role only once. |

## II. The Unified Service Catalog: Modeling Arc-i-Tech's Diverse Offerings

This section details the design of the website's "storefront." This schema must logically organize, manage, and present every service Arc-i-Tech offers. This is a significant architectural challenge, as the service portfolio is highly heterogeneous, spanning B2B project work 19, B2C educational courses 20, and various mentorship offerings. The design must be unified, scalable, and flexible.21

### A. The "Heterogeneous Service" Challenge

Arc-i-Tech's services are not like a simple e-commerce store's "products," which typically share a common set of attributes.23 The services are fundamentally different in their nature and metadata:

* A **"Software Development"** service has attributes like Tech\_Stack and Contract\_Type.
* A **"Project Mentorship"** service has attributes like Session\_Duration and Mentor\_Pool.
* A **"Mock Test"** service has attributes like Question\_Count and Time\_Limit.

Attempting to model this in a single, flat Services table with columns for *all* possible attributes would result in a massively sparse, unmanageable, and non-normalized database. Most columns would be NULL for any given service, which is highly inefficient and a clear violation of database design principles.5

### B. Analyzing and Rejecting Anti-Patterns

Two common but flawed patterns must be analyzed and rejected.

Anti-Pattern: Entity-Attribute-Value (EAV)

This model 26 would store attributes in a "vertical" format using tables like Service\_Attributes (service\_id, attribute\_name, attribute\_value). While this appears flexible, it is widely considered an anti-pattern, or "evil," in relational design.27 The reasons for this are critical:

1. **No Data Validation:** The attribute\_value column is almost always a generic VARCHAR or TEXT type. This means the system cannot enforce that Session\_Duration is a number or that Tech\_Stack is a valid entry. An attribute value could be "45 minutes" or "banana" with no way to stop it at the database level.27
2. **Query Complexity:** Retrieving the attributes for a single service requires complex "pivot" queries or multiple self-joins, which are notoriously slow and difficult to write and maintain.27
3. **Poor Performance:** The model scales poorly due to the complex joins and inability to properly index the "value" column.27

Anti-Pattern: One Giant Table

As previously mentioned, a single, wide table is a non-starter. It leads to massive data sparsity, is difficult to modify, and violates the foundational principles of normalization.5

### C. The Recommended "Hybrid Catalog" Architecture

The optimal solution is a hybrid architecture that combines the rigidity and integrity of a relational model with the flexibility of modern data types for non-essential metadata.

The Discriminator-Polymorphism Model

The core of this architecture is a central Service\_Catalog table. This table holds common, queryable, and universal data: name, description, base\_price, category, etc.

To solve the heterogeneity problem, we add a service\_type (ENUM) column. This column acts as a **discriminator**.20 The value in this column (e.g., 'PROJECT', 'COURSE', 'MENTORSHIP', 'ASSESSMENT', 'INTERNSHIP') tells the application *where to look for the rest of the service's data*.

This is a "polymorphic association" design.

* If service\_type = 'PROJECT', the application knows there is a corresponding row in the Projects table (see Section III) that holds the project-specific details.
* If service\_type = 'COURSE', the application knows there is a corresponding row in the Courses table (see Section IV-A) that holds the modules, lessons, etc.

This approach keeps the data fully normalized, type-safe, and performant. The *specifics* of a service (like its modules or tasks) are stored in their own dedicated, optimized, and relational tables.

Using JSONB for Display-Only Metadata

This polymorphic model is operationally perfect, but it creates a challenge for discovery. To build the public "Browse Services" page, the application would have to query the Service\_Catalog and then perform JOIN operations with five different tables (Projects, Courses, etc.) just to display a summary card for each service. This is highly inefficient.

The solution is to add a JSONB (or JSON) column to the Service\_Catalog table, as recommended by modern design patterns.29

* This column, named display\_attributes, will store a **denormalized, non-canonical** set of key-value pairs used *only for displaying the service in the catalog*.
* For a 'COURSE' service: {"duration": "6 Weeks", "level": "Beginner", "topics":}.
* For a 'MENTORSHIP' service: {"duration": "60 minutes", "format": "1-on-1 Video Call"}.

This hybrid approach 30 provides the best of both worlds: the "schema-less" flexibility of EAV 26 for *display purposes*, while retaining the full power, queryability, and data integrity of the relational model for the *core application logic*.

### D. Table Definitions

**Table: Service\_Categories**

* **Purpose:** To group services logically on the website (e.g., "Software Development," "Career Guidance").
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| category\_id | INT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the category. |
| name | VARCHAR(255) | NOT NULL | Name of the category. |
| description | TEXT | NULLABLE | Description shown on the category page. |
| parent\_category\_id | INT | NULLABLE, FOREIGN KEY (Service\_Categories.category\_id) | Self-referencing key for sub-categories. |

**Table: Service\_Catalog**

* **Purpose:** The central "master list" of all purchasable/engageable services.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| service\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the service. |
| name | VARCHAR(255) | NOT NULL | Public name of the service. |
| description | TEXT | NULLABLE | Detailed public description. |
| service\_type | ENUM | NOT NULL | Discriminator column: 'PROJECT', 'COURSE', 'MENTORSHIP', 'ASSESSMENT', 'INTERNSHIP'. |
| base\_price | DECIMAL(10, 2) | NOT NULL, DEFAULT 0.00 | The cost of the service. |
| is\_public | BOOLEAN | NOT NULL, DEFAULT true | Flag to show/hide from the public catalog. |
| display\_attributes | JSONB | NULLABLE | Flexible, non-canonical attributes for display. |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | When the service was added. |

**Table: Service\_Category\_Map (Junction Table)**

* **Purpose:** Allows a single service to appear in multiple categories (Many-to-Many).
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| service\_id | BIGINT | NOT NULL, FOREIGN KEY (Service\_Catalog.service\_id) |  |
| category\_id | INT | NOT NULL, FOREIGN KEY (Service\_Categories.category\_id) |  |
| *Composite* |  | PRIMARY KEY (service\_id, category\_id) |  |

## III. Schema for B2B Services: Project, Client, and Portfolio Management

This section details the data model for Arc-i-Tech's core B2B services, including "Software Development," "Desktop applications," "Android applications," "Software Consulting," and "Engineering Projects." This schema forms the backbone of a complete Project Management (PM) system designed to track engagements from inception to completion.34

### A. Modeling Clients and Projects

The system must first model the B2B clients (the companies) and the projects (the engagements) associated with them.38

**Table: Clients (Organizations)**

* **Purpose:** Stores information about the *companies* that hire Arc-i-Tech.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| client\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the client company. |
| client\_name | VARCHAR(255) | NOT NULL, UNIQUE | The legal name of the company. |
| website | VARCHAR(255) | NULLABLE | The company's official website. |
| address | TEXT | NULLABLE | Physical address of the company. |
| primary\_contact\_user\_id | BIGINT | NULLABLE, FOREIGN KEY (Users.user\_id) | A primary point of contact from the Users table. |

**Table: Projects**

* **Purpose:** The central entity for any B2B engagement. This table is the *target* for any service in the Service\_Catalog where service\_type = 'PROJECT'.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| project\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the project. |
| client\_id | BIGINT | NOT NULL, FOREIGN KEY (Clients.client\_id) | The company this project is for. |
| name | VARCHAR(255) | NOT NULL | The name of the project. |
| description | TEXT | NULLABLE | A detailed description of the project. |
| status | ENUM | NOT NULL | 'Planning', 'Active', 'On\_Hold', 'Completed'. |
| start\_date | DATE | NULLABLE | The official start date of the project. |
| end\_date | DATE | NULLABLE | The expected or actual completion date. |
| repo\_link | VARCHAR(255) | NULLABLE | Link to the project's code repository. |
| project\_manager\_user\_id | BIGINT | NULLABLE, FOREIGN KEY (Users.user\_id) | The internal Arc-i-Tech user managing this project. |

**Table: Project\_Users (Junction Table)**

* **Purpose:** Manages the Many-to-Many relationship between Users (both clients and internal staff) and Projects.
* A simple User\_Projects link is insufficient as it provides no context. We must store the *context* of the user's relationship to the project. This is a common requirement for "project-based roles".42 This table answers the question: *what is this user's role on this specific project?*
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| project\_user\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique ID for this relationship. |
| project\_id | BIGINT | NOT NULL, FOREIGN KEY (Projects.project\_id) |  |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) |  |
| project\_role | VARCHAR(100) | NOT NULL | e.g., 'Developer', 'Client\_Stakeholder', 'QA\_Tester'. |

### B. Modeling Project Tracking (Milestones, Tasks, Dependencies)

This part of the schema is essential for fulfilling the "one-click" project tracking requirement from the query. It must be structured to store the hierarchical breakdown of work and power visualizations like Gantt charts 43 and task boards.

**Table: Milestones**

* **Purpose:** Represents major project phases, deliverables, or "checkpoints" in the project lifecycle.37
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| milestone\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the milestone. |
| project\_id | BIGINT | NOT NULL, FOREIGN KEY (Projects.project\_id) | The project this milestone belongs to. |
| title | VARCHAR(255) | NOT NULL | Name of the milestone (e.g., "Phase 1: Deployment"). |
| description | TEXT | NULLABLE | Details of what this milestone signifies. |
| due\_date | DATE | NULLABLE | The target date for this milestone. |
| status | ENUM | NOT NULL | 'Pending', 'In\_Progress', 'Completed'. |

**Table: Tasks**

* **Purpose:** Stores the individual, actionable work items for the project.
* Projects are inherently hierarchical; a large Task (e.g., "Build user auth") can be broken into many Subtasks (e.g., "Design DB schema," "Write API endpoints").44 The most efficient and scalable way to model this is a *self-referencing foreign key* (an adjacency list). The parent\_task\_id column will link a subtask back to its parent Tasks.task\_id.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| task\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the task. |
| project\_id | BIGINT | NOT NULL, FOREIGN KEY (Projects.project\_id) | The project this task belongs to. |
| milestone\_id | BIGINT | NULLABLE, FOREIGN KEY (Milestones.milestone\_id) | Optional: which milestone this task is part of. |
| parent\_task\_id | BIGINT | NULLABLE, FOREIGN KEY (Tasks.task\_id) | Self-referencing key for subtasks. |
| title | VARCHAR(255) | NOT NULL | The title of the task. |
| description | TEXT | NULLABLE | Detailed description of the work. |
| assignee\_user\_id | BIGINT | NULLABLE, FOREIGN KEY (Users.user\_id) | The user responsible for this task. |
| status | ENUM | NOT NULL | 'Todo', 'In\_Progress', 'In\_Review', 'Done'. |
| start\_date | DATE | NULLABLE | When the task is planned to start. |
| due\_date | DATE | NULLABLE | When the task is due. |
| priority | ENUM | NOT NULL | 'Low', 'Medium', 'High', 'Critical'. |

**Table: Task\_Dependencies**

* **Purpose:** Links tasks to one another to define the project's "critical path" and prevent work from being done out of order.48
* It is crucial to store the *type* of dependency, as a simple link is ambiguous. The most common type is "Finish-to-Start" (Task B cannot start until Task A finishes).52
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| dependency\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the dependency. |
| predecessor\_task\_id | BIGINT | NOT NULL, FOREIGN KEY (Tasks.task\_id) | The task that must be completed first. |
| successor\_task\_id | BIGINT | NOT NULL, FOREIGN KEY (Tasks.task\_id) | The task that depends on the predecessor. |
| dependency\_type | ENUM | NOT NULL | 'FINISH\_TO\_START', 'START\_TO\_START', 'FINISH\_TO\_FINISH'. |

### C. Modeling the Public "Portfolio" Showcase

A critical distinction must be made between the *internal* project management system and the *public-facing* "showcase" portfolio. The internal Projects table is a sensitive, operational entity, containing all client information, internal notes, and task histories.54 The public showcase, as per the query, needs a *curated, sanitized, and marketing-approved* subset of this data to attract new clients.55

Exposing the Projects table directly to the website's front end would be a significant security and data-leak risk. Therefore, a separate, decoupled Portfolio\_Items table *must* be created. This table is populated by an Arc-i-Tech administrator, who can optionally link it to a completed project\_id for internal reference, but it only stores data that is safe for public consumption.

**Table: Portfolio\_Items**

* **Purpose:** To store the curated data for the public-facing "Our Work" section of the website.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| portfolio\_item\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the portfolio item. |
| project\_id | BIGINT | NULLABLE, FOREIGN KEY (Projects.project\_id) | *Optional* internal link to the completed project. |
| title | VARCHAR(255) | NOT NULL | The public, marketing-friendly title. |
| description | TEXT | NOT NULL | The public-facing case study or description. |
| hero\_image\_url | VARCHAR(255) | NULLABLE | URL for the main image. |
| technologies\_used | JSONB | NULLABLE | e.g., ``. |
| is\_featured | BOOLEAN | NOT NULL, DEFAULT false | Flag to feature this item on the homepage. |
| public\_url | VARCHAR(255) | NULLABLE | A link to the live project, if applicable. |
| date\_completed | DATE | NULLABLE | The public-facing completion date. |

## IV. Schema for B2C Services: Education, Mentorship, and Assessment

This section details the database architecture for Arc-i-Tech's Business-to-Consumer (B2C) services. This is a complex domain requiring four distinct sub-systems, each a standalone module that integrates with the central Service\_Catalog (Section II) and Users (Section I) tables.

### A. Learning Management Schema (Tutoring & Courses)

This schema is designed to manage "Technical competency building program (Tutor)" and any other structured courses. It follows established patterns for a Learning Management System (LMS).57

A key requirement for this system is handling "contextual enrollment." Research indicates that modeling just a User and a Course is insufficient.61 The "Tutor" service, in particular, implies multiple, distinct *instances* or *sections* of a course (e.g., "Java Tutoring - Mon 5PM w/ Mentor A," "Java Tutoring - Tues 7PM w/ Mentor B").62 Therefore, a User will not just enroll in a Course; they will enroll in a specific Course\_Section, which is tied to a specific tutor and schedule.

**Table Definitions:**

**Table: Courses**

* **Purpose:** The primary entity for a course, linked from the service catalog.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| course\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the course. |
| service\_id | BIGINT | NOT NULL, UNIQUE, FOREIGN KEY (Service\_Catalog.service\_id) | 1-to-1 link to the service catalog entry. |
| title | VARCHAR(255) | NOT NULL | The title of the course. |
| description | TEXT | NULLABLE | A detailed syllabus or description. |

**Table: Course\_Modules**

* **Purpose:** The hierarchical breakdown of a course into sections or modules.61
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| module\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the module. |
| course\_id | BIGINT | NOT NULL, FOREIGN KEY (Courses.course\_id) | The course this module belongs to. |
| title | VARCHAR(255) | NOT NULL | e.g., "Week 1: Introduction to React". |
| order\_index | INT | NOT NULL | The order in which modules are displayed. |

**Table: Course\_Lessons**

* **Purpose:** The individual content items (videos, articles, quizzes) within a module.61
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| lesson\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the lesson. |
| module\_id | BIGINT | NOT NULL, FOREIGN KEY (Course\_Modules.module\_id) | The module this lesson belongs to. |
| title | VARCHAR(255) | NOT NULL | e.g., "Understanding JSX". |
| content | TEXT | NULLABLE | The text-based content of the lesson. |
| video\_url | VARCHAR(255) | NULLABLE | Link to the lesson's video. |
| order\_index | INT | NOT NULL | The order of lessons within a module. |

**Table: Course\_Sections (for Tutoring)**

* **Purpose:** A specific, scheduled *instance* of a Course, led by a specific Tutor.62
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| section\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the section. |
| course\_id | BIGINT | NOT NULL, FOREIGN KEY (Courses.course\_id) | The course this section is an instance of. |
| tutor\_user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The mentor/tutor teaching this section. |
| schedule\_info | TEXT | NULLABLE | e.g., "Mondays & Wednesdays, 7PM-8PM EST". |
| max\_students | INT | NULLABLE | The enrollment cap for this section. |
| start\_date | DATE | NOT NULL | Start date of this section. |
| end\_date | DATE | NOT NULL | End date of this section. |

**Table: Course\_Enrollments (Junction Table)**

* **Purpose:** Links a User (student) to a Course\_Section (or directly to a Course if it is self-paced and has no sections).60
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| enrollment\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the enrollment. |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The student enrolled. |
| course\_id | BIGINT | NOT NULL, FOREIGN KEY (Courses.course\_id) | The course they are enrolled in. |
| section\_id | BIGINT | NULLABLE, FOREIGN KEY (Course\_Sections.section\_id) | The specific section, if applicable. |
| enrollment\_date | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | When the student enrolled. |
| progress | JSONB | NULLABLE | Stores progress, e.g., {"completed\_lessons": }. |
| grade | VARCHAR(10) | NULLABLE | Final grade or completion status. |

### B. Mentorship & Scheduling Schema (Guidance & Interviews)

This schema is designed to manage "Project Mentorship," "Mock interviews," and "Placement guidance." This is fundamentally an appointment booking system.66

The core architectural challenge here is matching mentor availability with mentee needs.70 A common but deeply flawed approach is to store availability in a non-normalized way, such as a text column (monday\_slots = "9,10,11").73 This is un-queryable, unmanageable, and a classic database design sin.

The *correct*, normalized model 73 is a Mentor\_Availabilities table where **each row represents a single, contiguous block of time** that a mentor is available (e.g., mentor\_id=5, start\_time='2025-10-20 09:00', end\_time='2025-10-20 12:00', status='OPEN'). When a mentee books a 1-hour slot (e.g., 10:00-11:00), the application logic is responsible for handling this: it might create the Appointment and then "split" the original availability block into two new ones (9-10 and 11-12).

Furthermore, some services like "Mock interviews" may be group sessions.75 Therefore, the Appointments table cannot simply have a single mentee\_id column. The Appointments table must represent the *event* itself (hosted by one mentor), and a separate Appointment\_Attendees junction table must be used to link *many* mentees to that single event.

**Table Definitions:**

**Table: Mentor\_Availabilities**

* **Purpose:** Stores the "bookable" time slots for mentors in a normalized, queryable format.73
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| availability\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for this time block. |
| mentor\_user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The mentor who is available. |
| start\_time | TIMESTAMP | NOT NULL | The start of the available block. |
| end\_time | TIMESTAMP | NOT NULL | The end of the available block. |
| status | ENUM | NOT NULL | 'OPEN', 'BOOKED' (or 'PARTIALLY\_BOOKED'). |

**Table: Appointments**

* **Purpose:** The central record of a booked meeting (the "event").
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| appointment\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the appointment. |
| service\_id | BIGINT | NOT NULL, FOREIGN KEY (Service\_Catalog.service\_id) | The service this appointment is for. |
| mentor\_user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The mentor hosting the appointment. |
| start\_time | TIMESTAMP | NOT NULL | The exact start time of the meeting. |
| end\_time | TIMESTAMP | NOT NULL | The exact end time of the meeting. |
| status | ENUM | NOT NULL | 'Scheduled', 'Completed', 'Cancelled'. |
| meeting\_link | VARCHAR(255) | NULLABLE | URL for the video call (e.g., Zoom, Meet). |
| notes | TEXT | NULLABLE | Notes from the mentor or mentee. |

**Table: Appointment\_Attendees (Junction Table)**

* **Purpose:** Links one or more mentees (Users) to a single Appointment to support group sessions.75
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| appointment\_id | BIGINT | NOT NULL, FOREIGN KEY (Appointments.appointment\_id) |  |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) |  |
| *Composite* |  | PRIMARY KEY (appointment\_id, user\_id) |  |

### C. Assessment Schema (Mock Tests)

This schema is designed to manage the "Mock Test" service. It requires a robust "Question Bank" architecture and a system for tracking user attempts and results.77

The question bank must be flexible. The normalized schema consists of Quizzes, Questions, and Answer\_Options.77 To support different types of questions (e.g., multiple choice, free text), the Questions table needs a question\_type column.80 The application logic will read this type and render the appropriate response UI. The Answer\_Options table (answer\_id, question\_id, answer\_text, is\_correct) is then used to populate the 'SINGLE\_CHOICE' and 'MULTIPLE\_CHOICE' question types.

For tracking results, it is critical to avoid non-normalized storage (like a JSON blob of answers).83 The correct, normalized approach 84 uses a "header/detail" model. The Quiz\_Attempts table (header) stores a record of the user taking the test (start, end, score). The User\_Answers table (detail) stores the specific answer given for *each* question in that attempt. This granular model is essential for analytics, such as determining which topics a student struggles with.87

**Table Definitions:**

**Table: Quizzes**

* **Purpose:** Defines a specific test or assessment.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| quiz\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the quiz. |
| service\_id | BIGINT | NOT NULL, UNIQUE, FOREIGN KEY (Service\_Catalog.service\_id) | 1-to-1 link to the service catalog. |
| title | VARCHAR(255) | NOT NULL | Title of the mock test. |
| time\_limit\_minutes | INT | NULLABLE | The duration of the test in minutes. |

**Table: Questions**

* **Purpose:** The master "question bank." A single question can exist here and be reused in multiple quizzes.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| question\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the question. |
| question\_text | TEXT | NOT NULL | The text of the question itself. |
| question\_type | ENUM | NOT NULL | 'SINGLE\_CHOICE', 'MULTIPLE\_CHOICE', 'FREE\_TEXT'. |
| topic | VARCHAR(100) | NULLABLE | Topic for analytics, e.g., 'Data Structures'.87 |

**Table: Quiz\_Questions (Junction Table)**

* **Purpose:** Links questions from the bank to a specific quiz (Many-to-Many).
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| quiz\_id | BIGINT | NOT NULL, FOREIGN KEY (Quizzes.quiz\_id) |  |
| question\_id | BIGINT | NOT NULL, FOREIGN KEY (Questions.question\_id) |  |
| order\_index | INT | NOT NULL | The number of this question in the quiz. |
| *Composite* |  | PRIMARY KEY (quiz\_id, question\_id) |  |

**Table: Answer\_Options**

* **Purpose:** Stores the pre-defined answer choices for multiple-choice questions.77
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| answer\_option\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for this choice. |
| question\_id | BIGINT | NOT NULL, FOREIGN KEY (Questions.question\_id) | The question this answer belongs to. |
| answer\_text | TEXT | NOT NULL | The text of the answer choice. |
| is\_correct | BOOLEAN | NOT NULL, DEFAULT false | Flag for the correct answer. |

**Table: Quiz\_Attempts (Header Table)**

* **Purpose:** A header record for a single instance of a user taking a quiz.83
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| attempt\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for this attempt. |
| quiz\_id | BIGINT | NOT NULL, FOREIGN KEY (Quizzes.quiz\_id) | The quiz that was taken. |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The user who took the quiz. |
| start\_time | TIMESTAMP | NOT NULL | When the user started the quiz. |
| end\_time | TIMESTAMP | NULLABLE | When the user finished or was cut off. |
| final\_score | DECIMAL(5, 2) | NULLABLE | The final calculated score. |

**Table: User\_Answers (Detail Table)**

* **Purpose:** Stores the specific answer(s) given by a user for each question during an attempt.87
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| user\_answer\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique ID for this answer. |
| attempt\_id | BIGINT | NOT NULL, FOREIGN KEY (Quiz\_Attempts.attempt\_id) | The attempt this answer belongs to. |
| question\_id | BIGINT | NOT NULL, FOREIGN KEY (Questions.question\_id) | The question that was answered. |
| answer\_option\_id | BIGINT | NULLABLE, FOREIGN KEY (Answer\_Options.answer\_option\_id) | The choice selected (if multiple choice). |
| free\_text\_answer | TEXT | NULLABLE | The text entered (if free text). |
| is\_correct | BOOLEAN | NULLABLE | Whether this answer was marked as correct. |

### D. Internship Program Schema (Certificates)

This schema is designed to manage the "Internship (Certificates)" service. This process involves an application and enrollment phase 88 followed by a period of managing intern work and assignments.91

A key architectural decision is how to model the *work* that interns perform. Interns will be given "backburner projects," "proposals," or "research tasks".91 Arc-i-Tech *already has* a complete, robust Project Management schema (defined in Section III) with Projects and Tasks.

It would be a significant mistake—introducing data redundancy and operational silos—to create a separate, parallel "Intern\_Projects" or "Intern\_Tasks" table.

The correct, efficient solution is to **reuse the existing Project Management system**. An "Internship Program" can be modeled as an internal Project. Once an intern's Intern\_Application is 'Approved', they are simply added to that project via the Project\_Users junction table and assigned Tasks from the Tasks table. This creates a seamless, unified system where interns are managed using the exact same tools as junior developers, providing them with a realistic work-tracking experience.

**Table Definitions:**

**Table: Internship\_Programs**

* **Purpose:** Defines a specific internship program or cohort.96
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| program\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the program. |
| service\_id | BIGINT | NOT NULL, UNIQUE, FOREIGN KEY (Service\_Catalog.service\_id) | 1-to-1 link to the service catalog. |
| title | VARCHAR(255) | NOT NULL | e.g., "Summer 2025 DevOps Internship". |
| description | TEXT | NOT NULL | Details of the program. |
| application\_start\_date | DATE | NOT NULL | When applications open. |
| application\_end\_date | DATE | NOT NULL | When applications close. |
| program\_start\_date | DATE | NOT NULL | When the internship begins. |
| program\_end\_date | DATE | NOT NULL | When the internship ends. |

**Table: Intern\_Applications**

* **Purpose:** To track a User's application to a specific Internship\_Program.90
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| application\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the application. |
| program\_id | BIGINT | NOT NULL, FOREIGN KEY (Internship\_Programs.program\_id) | The program being applied to. |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The user who is applying. |
| status | ENUM | NOT NULL | 'Pending', 'Under\_Review', 'Approved', 'Rejected'. |
| application\_data | JSONB | NULLABLE | Stores resume\_url, cover\_letter, etc. |
| submitted\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | When the application was submitted. |

**Table: Intern\_Assignments (Junction Table)**

* **Purpose:** Links an approved intern (User) to their assigned Tasks from the *existing* PM system.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| assignment\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique ID for this assignment. |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The intern. |
| task\_id | BIGINT | NOT NULL, FOREIGN KEY (Tasks.task\_id) | The task assigned (from Sec III). |
| assigned\_by\_user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The mentor/manager who assigned the task. |

**Table: Intern\_Journals**

* **Purpose:** To allow interns to submit "daily reports," "reflection assignments," or "progress reports" as required by many programs.89
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| journal\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the journal entry. |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The intern writing the entry. |
| program\_id | BIGINT | NOT NULL, FOREIGN KEY (Internship\_Programs.program\_id) | The program this entry is for. |
| entry\_date | DATE | NOT NULL | The date of the entry. |
| entry\_text | TEXT | NOT NULL | The content of the intern's journal. |
| feedback\_text | TEXT | NULLABLE | Feedback from a mentor on this entry. |
| feedback\_by\_user\_id | BIGINT | NULLABLE, FOREIGN KEY (Users.user\_id) | The mentor who provided feedback. |

## V. The Central Hub: Unifying Orders, Inquiries, and the User Dashboard

This final and most critical section details the "glue" that holds the entire, complex system together. It provides the architecture to fulfill the "one-click" project/software tracking requirement and models the business's lead-to-customer conversion pipeline.

### A. The "One-Click" Dashboard: A Master Order Junction

**The Challenge:** The core user query demands that a customer "can track their project/software program with one click." A user's dashboard must provide a "single pane of glass" showing *all* their engaged services: their active software project, their enrolled courses, and their upcoming mentorship appointments. Querying the 4-5 different "enrollment" tables (Project\_Users, Course\_Enrollments, Appointment\_Attendees, Intern\_Applications) to build this single dashboard view would be slow, complex, and unscalable.

The Solution: A "Unified Namespace" & Polymorphic Master Key

A "single, unified data ecosystem" is required.98 The solution is a "master junction table" 99 that functions as the user's master "order history" or list of "engaged services".23

We will create a master table named Service\_Orders.

When a user "buys," "registers," or "enrolls" in any service from the Service\_Catalog, a single row is created in this table. This row links the User to the Service\_Catalog item (e.g., user\_id=5, service\_id=10...).

This solves half the problem. The other half is: *how does this Service\_Orders row link to the specific Project, Course\_Enrollment, or Appointment that was generated by the purchase?*

The answer is a **polymorphic association**. The Service\_Orders table will contain two special columns:

1. instance\_id (e.g., 123)
2. instance\_type (e.g., 'PROJECT')

This (instance\_id, instance\_type) pair is a *polymorphic foreign key*. The instance\_type tells the application *which table* the instance\_id refers to.

**Resulting User Story (The "One-Click" Solution):**

1. A user logs in to their dashboard.
2. The application runs **one single, fast query**: SELECT \* FROM Service\_Orders WHERE user\_id = [current\_user].
3. The dashboard renders this list, showing all their services (Project Alpha, Java Course, Mock Interview).
4. The user clicks "Track" on "Project Alpha."
5. The application looks at that row's instance\_type ('PROJECT') and instance\_id (123).
6. It now knows exactly which "one-click" query to run: SELECT \* FROM Projects WHERE project\_id = 123.  
   This design is clean, scalable, and directly fulfills the primary user requirement.

**Table Definition:**

**Table: Service\_Orders (Master Junction Table)**

* **Purpose:** The central "master list" of all services a user has ever engaged with. This is the core of the user dashboard.
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| order\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for this order/engagement. |
| user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The user who owns this service. |
| service\_id | BIGINT | NOT NULL, FOREIGN KEY (Service\_Catalog.service\_id) | The catalog item that was ordered. |
| order\_status | ENUM | NOT NULL | 'Pending\_Payment', 'Active', 'Completed', 'Cancelled'. |
| price\_at\_purchase | DECIMAL(10, 2) | NOT NULL | The price the user agreed to pay. |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | The date of the order. |
| instance\_type | VARCHAR(100) | NOT NULL | Polymorphic type: 'PROJECT', 'COURSE\_ENROLLMENT', 'APPOINTMENT'. |
| instance\_id | BIGINT | NOT NULL | Polymorphic ID: links to Projects.project\_id, etc. |

### B. Lead and Inquiry Management (Simple CRM)

This schema models the "top-of-funnel" for the business, capturing and converting prospective clients and students.107 It requires two tables to model the "Inquiry -> Lead -> Customer" pipeline.110

**Table Definitions:**

**Table: Inquiries**

* **Purpose:** To capture raw, unstructured data from the website's main "Contact Us" form.110
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| inquiry\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the inquiry. |
| full\_name | VARCHAR(255) | NOT NULL | Name from the contact form. |
| email | VARCHAR(255) | NOT NULL | Email from the contact form. |
| phone | VARCHAR(50) | NULLABLE | Phone number from the contact form. |
| message | TEXT | NOT NULL | The user's message. |
| status | ENUM | NOT NULL | 'New', 'Read', 'Archived', 'Converted\_To\_Lead'. |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | When the form was submitted. |

**Table: Leads**

* **Purpose:** A "promoted" inquiry that has been qualified and is now an active sales lead for an Arc-i-Tech team member to track.108
* **Attributes:**

| **Column** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| lead\_id | BIGINT | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for the lead. |
| inquiry\_id | BIGINT | NULLABLE, FOREIGN KEY (Inquiries.inquiry\_id) | The original inquiry this lead came from. |
| lead\_name | VARCHAR(255) | NOT NULL | The qualified name of the lead. |
| email | VARCHAR(255) | NOT NULL | The lead's email. |
| status | ENUM | NOT NULL | 'New', 'Contacted', 'Proposal\_Sent', 'Won', 'Lost'. |
| assigned\_to\_user\_id | BIGINT | NOT NULL, FOREIGN KEY (Users.user\_id) | The internal user responsible for this lead. |
| notes | TEXT | NULLABLE | Internal notes on the lead's progress. |

Conversion Path:

This two-table schema models the entire business funnel.

1. A user fills out the contact form, creating a row in Inquiries.
2. An admin reviews the inquiry and "converts" it, creating a new row in Leads and updating the Inquiries.status to 'Converted\_To\_Lead'.
3. The sales team works the Lead.
4. When the Leads.status is set to 'Won', the application logic triggers the final conversion:  
   a. A new User is created (if one doesn't exist).  
   b. A new Client is created (if B2B).  
   c. A new Project (or other service instance) is created.  
   d. A new Service\_Order is created, linking the User, Service\_Catalog item, and the new instance\_id.  
   This completes the data lifecycle from anonymous visitor to paying customer.

## VI. Concluding Architectural Synopsis and Entity-Relationship Diagram (ERD)

### A. Architectural Summary

The database architecture detailed in this report provides a comprehensive, scalable, and normalized **Logical Schema** 113 designed to meet all of Arc-i-Tech's current and future needs. The design is built upon established, high-performance data modeling principles, ensuring the final system is maintainable, flexible, and robust.

The core principles of this architecture are:

1. **Identity & Access Decoupling:** The "Composition over Inheritance" 7 model for user management is the most significant architectural feature. It correctly separates a user's *Identity* (the Users table), their *Data* (the various ...\_Profiles tables), and their *Behavior* (the RBAC schema).4 This allows for a flexible, multi-role system where a single user can be a client, a student, and a mentor simultaneously without data duplication.
2. **Hybrid Service Catalog:** The design for the Service\_Catalog 21 explicitly rejects the "EAV is evil" anti-pattern.27 Instead, it uses a hybrid approach: a service\_type *discriminator* column points to fully normalized, relational tables (e.g., Projects, Courses) for core logic, while a flexible JSONB display\_attributes column provides schema-less display metadata.30
3. **Unified User Dashboard:** The "one-click" tracking requirement is solved via the polymorphic, "master order" table (Service\_Orders). This provides a "single pane of glass" 98 for all user-service interactions, creating a performant and simple-to-query dashboard that unifies all of Arc-i-Tech's disparate service offerings.
4. **Normalized & Reusable:** The entire design adheres to 3rd Normal Form 8 by separating concerns (e.g., Questions from Quizzes, Tasks from Projects), which avoids data redundancy and modification anomalies.25 It promotes reusability, most notably by leveraging the core Project Management schema to also manage the Internship Program, reducing development time and unifying the user experience.

### B. Entity-Relationship Diagram (ERD) - Textual Description

This section provides a textual description of the primary relationships between the core entities, serving as a blueprint for implementation.113

**1. Foundation: Users & Access (Section I)**

* Users (PK: user\_id)
  + Has a 1-to-1 relationship with Student\_Profiles (FK: user\_id).
  + Has a 1-to-1 relationship with Client\_Profiles (FK: user\_id).
  + Has a 1-to-1 relationship with Mentor\_Profiles (FK: user\_id).
  + Has a Many-to-Many relationship with Roles via User\_Roles (FKs: user\_id, role\_id).
* Roles (PK: role\_id)
  + Has a Many-to-Many relationship with Permissions via Role\_Permissions (FKs: role\_id, permission\_id).
* Permissions (PK: permission\_id)

**2. The Funnel: Leads & Clients (Section V-B & III-A)**

* Inquiries (PK: inquiry\_id)
  + Has a 1-to-1 relationship with Leads (FK: inquiry\_id).
* Leads (PK: lead\_id)
  + Has a 1-to-Many relationship with Users (FK: assigned\_to\_user\_id).
* Clients (PK: client\_id)
  + Has a 1-to-Many relationship with Users (FK: primary\_contact\_user\_id).
  + Client\_Profiles (from Section I) links to Clients (FK: client\_id).

**3. The Catalog: Services (Section II)**

* Service\_Categories (PK: category\_id)
  + Has a 1-to-Many relationship with itself (FK: parent\_category\_id).
* Service\_Catalog (PK: service\_id)
  + Has a Many-to-Many relationship with Service\_Categories via Service\_Category\_Map (FKs: service\_id, category\_id).

**4. The Service Implementations (Sections III & IV)**

* Service\_Catalog has a 1-to-1 polymorphic relationship with:
  + Projects (FK: service\_id - *Note: The outline implies Projects are generated from B2B clients, not always a 1-to-1 catalog item. The final design connects Projects to Clients, and the order links them via the polymorphic key.*)
  + Courses (FK: service\_id).
  + Quizzes (FK: service\_id).
  + Internship\_Programs (FK: service\_id).
  + Appointments (FK: service\_id).

**5. Implementation Details: B2B Projects (Section III)**

* Projects (PK: project\_id)
  + Links to Clients (FK: client\_id).
  + Links to Users (FK: project\_manager\_user\_id).
  + Has a Many-to-Many relationship with Users via Project\_Users (FKs: project\_id, user\_id).
  + Has a 1-to-Many relationship with Milestones (FK: project\_id).
  + Has a 1-to-Many relationship with Tasks (FK: project\_id).
* Milestones (PK: milestone\_id)
  + Has a 1-to-Many relationship with Tasks (FK: milestone\_id).
* Tasks (PK: task\_id)
  + Links to Users (FK: assignee\_user\_id).
  + Has a 1-to-Many relationship with itself (FK: parent\_task\_id).
  + Has a Many-to-Many relationship with itself via Task\_Dependencies (FKs: predecessor\_task\_id, successor\_task\_id).
* Portfolio\_Items (PK: portfolio\_item\_id)
  + Has an *optional* 1-to-1 relationship with Projects (FK: project\_id).

**6. Implementation Details: B2C Services (Section IV)**

* **LMS:**
  + Courses (PK: course\_id)
    - Has a 1-to-Many relationship with Course\_Modules (FK: course\_id).
    - Has a 1-to-Many relationship with Course\_Sections (FK: course\_id).
  + Course\_Modules (PK: module\_id) -> Course\_Lessons (FK: module\_id).
  + Course\_Sections (PK: section\_id) -> Users (FK: tutor\_user\_id).
  + Course\_Enrollments (PK: enrollment\_id) links Users, Courses, and Course\_Sections.
* **Mentorship:**
  + Mentor\_Availabilities (PK: availability\_id) -> Users (FK: mentor\_user\_id).
  + Appointments (PK: appointment\_id) -> Users (FK: mentor\_user\_id).
  + Appointments has a Many-to-Many relationship with Users via Appointment\_Attendees (FKs: appointment\_id, user\_id).
* **Assessment:**
  + Quizzes (PK: quiz\_id)
    - Has a Many-to-Many relationship with Questions via Quiz\_Questions (FKs: quiz\_id, question\_id).
  + Questions (PK: question\_id)
    - Has a 1-to-Many relationship with Answer\_Options (FK: question\_id).
  + Quiz\_Attempts (PK: attempt\_id) links Quizzes and Users.
  + User\_Answers (PK: user\_answer\_id) links Quiz\_Attempts, Questions, and Answer\_Options.
* **Internship:**
  + Internship\_Programs (PK: program\_id)
    - Has a 1-to-Many relationship with Intern\_Applications (FK: program\_id).
  + Intern\_Applications (PK: application\_id) -> Users (FK: user\_id).
  + Intern\_Assignments (PK: assignment\_id) links Users and Tasks.
  + Intern\_Journals (PK: journal\_id) links Users and Internship\_Programs.

**7. The Unifying Hub: Master Orders (Section V-A)**

* Service\_Orders (PK: order\_id)
  + This is the central hub.
  + It links to Users (FK: user\_id).
  + It links to Service\_Catalog (FK: service\_id).
  + It has a *polymorphic* link (instance\_id, instance\_type) to the primary keys of Projects, Course\_Enrollments, Appointments, Quiz\_Attempts, and Intern\_Applications.

#### Works cited

1. Design of the database with multiple user roles - sql - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/27625860/design-of-the-database-with-multiple-user-roles>
2. Choosing Between Database Tables — Separate Vs Single for Admins and Users, accessed November 6, 2025, <https://muwangaxyz.medium.com/choosing-between-database-tables-separate-vs-single-for-admins-and-users-97673e8de457>
3. Database design: 3 types of users, separate or one table? [closed] - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/8479252/database-design-3-types-of-users-separate-or-one-table>
4. Modelling a database structure for multiple user types and their contact information, accessed November 6, 2025, <https://dba.stackexchange.com/questions/183119/modelling-a-database-structure-for-multiple-user-types-and-their-contact-informa>
5. Database design - single subtype in a separate table vs common table - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/69941370/database-design-single-subtype-in-a-separate-table-vs-common-table>
6. tables of each user type or just one users table - laravel - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/laravel/comments/xwyvdg/tables_of_each_user_type_or_just_one_users_table/>
7. Composition vs Inheritance | DigitalOcean, accessed November 6, 2025, <https://www.digitalocean.com/community/tutorials/composition-vs-inheritance>
8. Composition Over Inheritance Table Structure : r/SoftwareEngineering - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/SoftwareEngineering/comments/1iba5pg/composition_over_inheritance_table_structure/>
9. Database design: 4 types of users but have different functionality , separate or one table?, accessed November 6, 2025, <https://softwareengineering.stackexchange.com/questions/261137/database-design-4-types-of-users-but-have-different-functionality-separate-or>
10. Data model - inheritance - Software Engineering Stack Exchange, accessed November 6, 2025, <https://softwareengineering.stackexchange.com/questions/251527/data-model-inheritance>
11. Joined Table Inheritance Mapping vs Composition in Object Relational Databases, accessed November 6, 2025, <https://stackoverflow.com/questions/25429558/joined-table-inheritance-mapping-vs-composition-in-object-relational-databases>
12. Building the optimal user database model for your application - DoneDone, accessed November 6, 2025, <https://www.donedone.com/blog/building-the-optimal-user-database-model-for-your-application>
13. Designing a Role-Based Access Control (RBAC) System: A Scalable Approach - Medium, accessed November 6, 2025, <https://medium.com/@07rohit/designing-a-role-based-access-control-rbac-system-a-scalable-approach-441f05168933>
14. How to Design an RBAC (Role-Based Access Control) System - NocoBase, accessed November 6, 2025, <https://www.nocobase.com/en/blog/how-to-design-rbac-role-based-access-control-system>
15. How RBAC Works: The Fundamentals of Role-Based Access Control - CelerData, accessed November 6, 2025, <https://celerdata.com/glossary/role-based-access-control-rbac>
16. mysql - DB schema for RBAC with multiple levels of roles - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/7329150/db-schema-for-rbac-with-multiple-levels-of-roles>
17. how to design database for Access Control Privileges ? : r/node - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/node/comments/14x77lc/how_to_design_database_for_access_control/>
18. User with multiple roles and multiple teams database design - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/64595033/user-with-multiple-roles-and-multiple-teams-database-design>
19. B2B eCommerce Catalog: Everything You Need to Know - Net Solutions, accessed November 6, 2025, <https://www.netsolutions.com/hub/b2b-ecommerce/catalog/>
20. Design a world-class service catalog - ServiceNow, accessed November 6, 2025, <https://www.servicenow.com/community/s/cgfwn76974/attachments/cgfwn76974/developer-forum/1132780/1/service-catalog-design-sn.pdf>
21. IT Service Catalogs: Best Practices and Integration Tips | Atlassian, accessed November 6, 2025, <https://www.atlassian.com/itsm/service-request-management/service-catalog>
22. Service Catalogs: Defining Standardized Database Services - Oracle, accessed November 6, 2025, <https://www.oracle.com/technetwork/database/database-cloud/private/service-catalogs-for-dbaas-2041214.pdf>
23. How to Design a Relational Database for E-commerce Website - GeeksforGeeks, accessed November 6, 2025, <https://www.geeksforgeeks.org/dbms/how-to-design-a-relational-database-for-e-commerce-website/>
24. Ecommerce Database Design: ER Diagram for Online Shopping - Redgate Software, accessed November 6, 2025, <https://www.red-gate.com/blog/er-diagram-for-online-shop>
25. Database Design: Multiple tables vs a single table - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/9021838/database-design-multiple-tables-vs-a-single-table>
26. Entity–attribute–value model - Wikipedia, accessed November 6, 2025, <https://en.wikipedia.org/wiki/Entity%E2%80%93attribute%E2%80%93value_model>
27. How to design a database schema for multiple user types?, accessed November 6, 2025, <https://dba.stackexchange.com/questions/323789/how-to-design-a-database-schema-for-multiple-user-types>
28. database schema for products attributes - mysql - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/14194841/database-schema-for-products-attributes>
29. Database Design for Product Management | by Mojtaba Azad - Medium, accessed November 6, 2025, <https://medium.com/@pesarakex/database-design-for-product-management-9280fd7c66fe>
30. Designing Product Catalogs in SQL Server using JSON - Microsoft Community Hub, accessed November 6, 2025, <https://techcommunity.microsoft.com/t5/sql-server-blog/designing-product-catalogs-in-sql-server-using-json/ba-p/384594>
31. Offsite Retargeting of Unauthenticated Visitors | Adobe Real-Time Customer Data Platform, accessed November 6, 2025, <https://experienceleague.adobe.com/en/docs/project-aim-demo/rtcdp/intro/rtcdpb2b-intro/12-b2b-step-1-model-data>
32. From Complexity to Clarity: Organizing Variant-Wise Product Storage for Your Online Store, accessed November 6, 2025, <https://mediusware.com/blog/from-complexity-to-clarity-organizing-variant-wise>
33. Storing Dynamic Attributes - Sparse Columns, EAV, and JSONB Explained | Leapcell, accessed November 6, 2025, <https://leapcell.io/blog/storing-dynamic-attributes-sparse-columns-eav-and-jsonb-explained>
34. How to Design ER Diagrams for Project Management Software - GeeksforGeeks, accessed November 6, 2025, <https://www.geeksforgeeks.org/sql/how-to-design-er-diagrams-for-project-management-software/>
35. What's a Project Management Database? | ClickUp, accessed November 6, 2025, <https://clickup.com/blog/project-management-database/>
36. A Database Schema for Engineering Project Management | by Roshn Noronha - Medium, accessed November 6, 2025, <https://roshnnoronha.medium.com/a-database-schema-for-engineering-project-management-51d1b1f1078c>
37. Database schema for project management app : r/learnprogramming - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/learnprogramming/comments/1dp5fm4/database_schema_for_project_management_app/>
38. Database design, development & consulting services - Aristek Systems, accessed November 6, 2025, <https://aristeksystems.com/services/database-design-development/>
39. Database Consulting Services - Percona, accessed November 6, 2025, <https://www.percona.com/services/consulting>
40. Project Tracking Database - Efficient Project Management - BigTime Software, accessed November 6, 2025, <https://www.bigtime.net/blogs/project-tracking-database/>
41. A project management system for your design team that connects all your work - Notion, accessed November 6, 2025, <https://www.notion.com/help/guides/a-project-management-system-for-your-design-team-that-connects-all-your-work>
42. Database design for project-based roles - postgresql - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/68170543/database-design-for-project-based-roles>
43. A Gantt Chart Guide with Definitions & Examples - ProjectManager, accessed November 6, 2025, <https://www.projectmanager.com/guides/gantt-chart>
44. Gantt Chart: A Complete Guide (Definition & Examples) - Canva, accessed November 6, 2025, <https://www.canva.com/online-whiteboard/gantt-chart/>
45. Gantt chart guide: How to create and use one - Atlassian, accessed November 6, 2025, <https://www.atlassian.com/agile/project-management/gantt-chart>
46. What is a Gantt chart? A guide to mapping your project timelines - Asana, accessed November 6, 2025, <https://asana.com/resources/gantt-chart-basics>
47. Project Tracking Database: The Complete Guide with Types & Tools | Lark, accessed November 6, 2025, <https://www.larksuite.com/en_us/blog/project-tracking-database>
48. Project Management Software Database Structure and Schema, accessed November 6, 2025, <https://databasesample.com/database/project-management-software-database>
49. Break tasks into manageable steps with sub-tasks and dependencies - Notion, accessed November 6, 2025, <https://www.notion.com/help/guides/tasks-manageable-steps-sub-tasks-dependencies>
50. Easier ways to work with Subtasks that are within the same table, Grouping, accessed November 6, 2025, <https://community.coda.io/t/easier-ways-to-work-with-subtasks-that-are-within-the-same-table-grouping/13714>
51. How to create and use a milestone chart for project management - Atlassian, accessed November 6, 2025, <https://www.atlassian.com/work-management/project-management/project-planning/milestone-chart>
52. Dependencies in Project Management: Types, Tools, and Tactics, accessed November 6, 2025, <https://thedigitalprojectmanager.com/productivity/task-dependencies/>
53. A guide to functional dependencies in database systems - Wrike, accessed November 6, 2025, <https://www.wrike.com/blog/functional-dependencies-database-systems/>
54. Advice on database design for portfolio website - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/14942981/advice-on-database-design-for-portfolio-website>
55. Setting Up the Database for Your Portfolio Website : Day 3 - Compalgo Labs, accessed November 6, 2025, <https://compalgolabs.com/setting-up-the-database/>
56. faizanfareed/Personal-portfolio-database-design-ERD - GitHub, accessed November 6, 2025, <https://github.com/faizanfareed/Personal-portfolio-database-design-ERD->
57. LMS Database Schema | PDF - Scribd, accessed November 6, 2025, <https://www.scribd.com/document/902336935/LMS-Database-Schema>
58. How to Design a Database for Learning Management System (LMS) - GeeksforGeeks, accessed November 6, 2025, <https://www.geeksforgeeks.org/sql/how-to-design-a-database-for-learning-management-system-lms/>
59. Learning Management System Database Structure and Schema, accessed November 6, 2025, <https://databasesample.com/database/learning-management-system-database>
60. How to Design a Database for Online Learning Platform - GeeksforGeeks, accessed November 6, 2025, <https://www.geeksforgeeks.org/sql/how-to-design-a-database-for-online-learning-platform/>
61. Database Design for a Learning Management System - Redgate Software, accessed November 6, 2025, <https://www.red-gate.com/blog/database-design-management-system>
62. Tutors, Students, Courses database design - mysql - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/49341255/tutors-students-courses-database-design>
63. A Guide to Building a Free Database for Education in 11 Simple Steps - Knack, accessed November 6, 2025, <https://www.knack.com/blog/database-for-education/>
64. Database Design for an Online Course Website - YouTube, accessed November 6, 2025, <https://www.youtube.com/watch?v=FZVHZTaot1E>
65. Designing a Comprehensive Database for Higher Education | by Hammad Ali Shah, accessed November 6, 2025, <https://medium.com/@hammadalishah01/designing-a-comprehensive-database-for-higher-education-4ca3a1d7bcfc>
66. Structuring An Effective Mentorship Program: 6-Steps to Success, accessed November 6, 2025, <https://www.qooper.io/blog/the-ultimate-guide-to-structuring-an-effective-mentorship-program>
67. A Database Model to Manage Appointments and Organize Schedules - Redgate Software, accessed November 6, 2025, <https://www.red-gate.com/blog/a-database-model-to-manage-appointments-and-organize-schedules>
68. DB Schema for Appointment Booking App: What is the correct relationship between Doctors, Appointments, TimeSlots, Patients? - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/25563459/db-schema-for-appointment-booking-app-what-is-the-correct-relationship-between>
69. Database schema for Timegrid - DrawSQL, accessed November 6, 2025, <https://drawsql.app/templates/timegrid>
70. Mentor Matching Made Easy with Software | Chronus, accessed November 6, 2025, <https://chronus.com/software/mentoring-software/mentor-matching>
71. A Guide To Scaling A Successful Mentoring Programme By PLD, accessed November 6, 2025, <https://www.pldmentoring.com/blog/scaling-a-mentoring-programme>
72. Mentorship schedule matchmaking algorithm - Software Engineering Stack Exchange, accessed November 6, 2025, <https://softwareengineering.stackexchange.com/questions/418268/mentorship-schedule-matchmaking-algorithm>
73. Storing timetable availability in SQL table - database - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/19783680/storing-timetable-availability-in-sql-table>
74. Creating a mentoring platform - Airtable Community, accessed November 6, 2025, <https://community.airtable.com/fid-13/tid-7998>
75. What is Group Mentoring? A Complete Guide | Guider AI, accessed November 6, 2025, <https://guider-ai.com/blog/what-is-group-mentoring-a-complete-guide/>
76. How to Prepare Your Members for an Awesome Mentoring Session - Wild Apricot, accessed November 6, 2025, <https://www.wildapricot.com/blog/mentoring-session>
77. Database design for an online quiz - sql - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/19056865/database-design-for-an-online-quiz>
78. Database Schema for a multiplayer quiz game - Software Engineering Stack Exchange, accessed November 6, 2025, <https://softwareengineering.stackexchange.com/questions/385206/database-schema-for-a-multiplayer-quiz-game>
79. How to store quiz questions in db : r/Database - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/Database/comments/ah6662/how_to_store_quiz_questions_in_db/>
80. Can you recommend a database design for quiz questions and answers that would allow for an extensible number of question types? - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/4270776/can-you-recommend-a-database-design-for-quiz-questions-and-answers-that-would-al>
81. Schema database for quiz answer - Laracasts, accessed November 6, 2025, <https://laracasts.com/discuss/channels/eloquent/schema-database-for-quiz-answer>
82. Important DBMS MCQ (Multiple Choice Questions) - InterviewBit, accessed November 6, 2025, <https://www.interviewbit.com/dbms-mcq/>
83. Optimal way to store user test score data? : r/learnprogramming - Reddit, accessed November 6, 2025, <https://www.reddit.com/r/learnprogramming/comments/1g8gsj8/optimal_way_to_store_user_test_score_data/>
84. Database design for an exam - sql server - DBA Stack Exchange, accessed November 6, 2025, <https://dba.stackexchange.com/questions/142282/database-design-for-an-exam>
85. best performance database design to store students exams result - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/23418534/best-performance-database-design-to-store-students-exams-result>
86. Addition of Test and Grade Information to Student Database - Microsoft Learn, accessed November 6, 2025, <https://learn.microsoft.com/en-us/answers/questions/4832127/addition-of-test-and-grade-information-to-student>
87. Database design for tracking student exam performance at topic level with possibly 10 billion rows, accessed November 6, 2025, <https://dba.stackexchange.com/questions/327751/database-design-for-tracking-student-exam-performance-at-topic-level-with-possib>
88. How to Get an Internship as a Database Administrator - GeeksforGeeks, accessed November 6, 2025, <https://www.geeksforgeeks.org/work-experiences/how-to-get-an-internship-as-a-database-administrator/>
89. Design and implementation of web-based internship information system at vocational school, accessed November 6, 2025, <https://www.researchgate.net/publication/329401975_Design_and_implementation_of_web-based_internship_information_system_at_vocational_school>
90. Design For Internship and Career | PDF | Databases - Scribd, accessed November 6, 2025, <https://www.scribd.com/document/736849675/1714225940266-Design-for-Internship-and-Career>
91. Examples of Internship Projects, accessed November 6, 2025, <https://admin.ks.gov/media/cms/Example_of_Project_Ideas_20211230_C51C62FCD1F6C.pdf>
92. SAMPLE INTERNSHIP PROJECTS OF VALUE, accessed November 6, 2025, <https://nc01911451.schoolwires.net/cms/lib/NC01911451/Centricity/Domain/3962/Sample%20Intern%20Projects.pdf>
93. Internships – Sample Projects - UW Career Center - University of Washington, accessed November 6, 2025, <https://careers.uw.edu/resources/sample-intern-projects/>
94. School of Business Internship Program Internship Assignment Instructions Business students who enroll in an internship for acade, accessed November 6, 2025, <https://www.eiu.edu/business/files/Internship_Assignment_Instructions_Complete_FA19.pdf>
95. Sample Syllabus for Internships - Pepperdine Community, accessed November 6, 2025, <https://community.pepperdine.edu/seaver/dean/faculty/internships/internship-syllabus.htm>
96. theagileturtles/internship-management-app - GitHub, accessed November 6, 2025, <https://github.com/theagileturtles/internship-management-app>
97. Internship Program Management Information System with Lean Management, accessed November 6, 2025, <https://www.ijiet.org/vol12/1580-IJIET-2906.pdf>
98. Enabling Digital Work Order Management with a Unified Namespace (UNS) - HiveMQ, accessed November 6, 2025, <https://www.hivemq.com/blog/enabling-digital-work-order-management-unified-namespace-uns/>
99. Map many-to-many relationships (Visual Database Tools) - Microsoft Learn, accessed November 6, 2025, <https://learn.microsoft.com/en-us/ssms/visual-db-tools/map-many-to-many-relationships-visual-database-tools>
100. What is the point of a junction table - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/70609439/what-is-the-point-of-a-junction-table>
101. Is it a one-to-many or many-to-many relationship? - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/78773066/is-it-a-one-to-many-or-many-to-many-relationship>
102. What do you call tables that create many to many relationships? - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/5996368/what-do-you-call-tables-that-create-many-to-many-relationships>
103. Create a Junction Table for Many-to-Many Relationships in SQL - w3resource, accessed November 6, 2025, <https://www.w3resource.com/sql-exercises/sql-query-to-design-a-junction-table-for-a-many-to-many-relationship.php>
104. Database schema for an online shop - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/35612778/database-schema-for-an-online-shop>
105. Building an E-commerce Database Schema with SQL | by Dawson Brookes | Medium, accessed November 6, 2025, <https://medium.com/@dedahbrookes/building-an-e-commerce-database-schema-with-sql-50933937fe18>
106. How to model a relational table that store order details - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/64108825/how-to-model-a-relational-table-that-store-order-details>
107. A Lead Management Pipeline by Crmble - Trello, accessed November 6, 2025, <https://trello.com/templates/business/a-lead-management-pipeline-by-crmble-WyPtJ3YS>
108. 14 Free Lead Management Templates (Full Review) - OnePageCRM, accessed November 6, 2025, <https://www.onepagecrm.com/blog/lead-management-templates/>
109. A CRM Database: The Tables & Fields Behind the UI - CRM Switch, accessed November 6, 2025, <https://crmswitch.com/crm/crm-database/>
110. Database Design for Customer Support Systems - GeeksforGeeks, accessed November 6, 2025, <https://www.geeksforgeeks.org/dbms/database-design-for-customer-support-systems/>
111. MySQL / Lead Management / Campaign - Structure [closed] - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/32384694/mysql-lead-management-campaign-structure>
112. Whats the best relational database structure for this simple contact form? - Stack Overflow, accessed November 6, 2025, <https://stackoverflow.com/questions/53606055/whats-the-best-relational-database-structure-for-this-simple-contact-form>
113. 4 Database Schema Examples for Various Applications | Airbyte, accessed November 6, 2025, <https://airbyte.com/data-engineering-resources/database-schema-examples>
114. Complete Guide to Database Schema Design - Integrate.io, accessed November 6, 2025, <https://www.integrate.io/blog/complete-guide-to-database-schema-design-guide/>
115. What Is a Database Schema? - IBM, accessed November 6, 2025, <https://www.ibm.com/think/topics/database-schema>
116. How to Design a Database Schema | Miro, accessed November 6, 2025, <https://miro.com/diagramming/how-to-design-database-schema/>
117. MySQL Workbench: Visual Database Design, accessed November 6, 2025, <https://www.mysql.com/products/workbench/design/>