LAB: DATABASE DESIGN AND BUILD PART 1

1) Tables to build:

Response:

- Courses
- LearningOutcomes
- Instructors
- CourseOfferings
- InstructorAssignments

Explanation:
The Courses table serves as the "base" of the database. It denotes the core curriculum offerings. Through Courses, we store essential information about each course independent of the semester during which it's offered or the instructor.

The LearningOutcomes table is separated from Courses to accommodate the fact that courses can have multiple learning outcomes, and these outcomes may change over time. This separation allows for greater flexibility in managing and updating learning outcomes without affecting the core course data.

The Instructors table stores information about the faculty members who teach the courses. By separating this from course information, we can easily manage instructor data and allow for scenarios where instructors teach multiple courses or courses have multiple instructors.

The CourseOfferings table represents specific instances of courses being offered in particular semester-year pairs. This allows for the same course to be offered multiple times with potentially different instructors/time periods.

The InstructorAssignments table serves as a mapping table. It creates a many-to-many relationship between CourseOfferings and Instructors. This allows for multiple instructors to be assigned to a single course offering, and for instructors

to be assigned to multiple course offerings.

2) Primary keys for each table:

Response:

- Courses: CourseID (e.g., auto-incrementing integer or unique course code)
- LearningOutcomes: LOID (auto-incrementing integer)
- Instructors: InstructorID (auto-incrementing integer)
- CourseOfferings: OfferingID (auto-incrementing integer)
- InstructorAssignments: AssignmentID (auto-incrementing integer)

Explanation:
The motivation for the primary key choices are the need for efficient and unique inter-table indexing.

For the Courses table, use of the auto-incrementing CourseID as the PK allows for easy referencing and joins with other tables. Technically speaking, a unique course code could serve an equivalent purpose. However, since course titles and id's can occasionally change with time, using an auto-intcrementing integer gives better long-term stability.

The LearningOutcomes, Instructors, CourseOfferings, and InstructorAssignments tables all use auto-incrementing integer PKs (e.g. LOID, InstructorID, OfferingID, and AssignmentID). There are four main advantages to this approach:

I. Each record is guaranteed to have a unique identifier.

II. It allows for efficient indexing and faster joins between tables.

III. It provides a stable reference point even if other attributes of the record change.

IV. We can readily generate new unique values when we insert new records.

3) Foreign keys for each table:

Response:

- LearningOutcomes: CourseID (references Courses table)
- CourseOfferings: CourseID (references Courses table)
- InstructorAssignments: (Compound Key)

- OfferingID (references CourseOfferings table)
- InstructorID (references Instructors table)

Explanation:
The chosen foreign key structure seeks to ensure referential integrity and efficient multi-table query of data.

For LearningOutcomes, the CourseID foreign key links each learning outcome to its associated course. This lets us retrieve all learning outcomes for a given course. Moreover, we can readily find that course to which a particular LearningOutcome belongs.

For CourseOfferings, the CourseID foreign key links each offering to its corresponding course in the Courses table. This relationship allows us to find all offerings of a particular course or to get course details for a specific offering.

The InstructorAssignments table uses a compound foreign key, referencing both the CourseOfferings and Instructors tables. This structure creates a many-to-many relationship between course offerings and instructors. The choice of compound key was motivated by two foreseeable scenarios:

I. A single course offering can have multiple instructors

II. An instructor can be assigned to multiple course offerings

This foreign key structure ensures data integrity by preventing orphaned records (e.g., you can't have a course offering without a corresponding course, or an instructor assignment without both a valid course offering and a valid instructor). It also facilitates complex queries that can traverse these relationships to answer questions about courses, their offerings, learning outcomes, and instructor assignments.

4) Active/Inactive flags:

Response: (IsActive flag)
We'll add a boolean field "IsActive" to the tables:

- 4.a) Courses table: IsActive (boolean)
- 4.b) Instructors table: IsActive (boolean)
- 4.c) LearningOutcomes table: IsActive (boolean)

Explanation:
The addition of an "IsActive" boolean field to the Courses, Instructors, and

LearningOutcomes tables stems from the need for tracking current and historical data without deletion.

For the Courses table, the IsActive flag indicates whether a course is offered or retired from the curriculum. This preserves historical data for courses no longer active. Ergo, record values are held for long-term analysis and record-keeping.

In the Instructors table, the IsActive flag denotes whether an instructor is employed or not. This maintains a record of all instructors associated with the program, while allowing filtering for current faculty members.

For the LearningOutcomes table, the IsActive flag enables tracking of changes in course objectives over time. As curricula evolve, we anticipate that learning outcomes may update or change. Using an IsActive flag instead of deleting old outcomes maintains a historical record of course goal changes, which proves valuable for accreditation purposes or curriculum review.

Use of the aforementioned IsActive flag system seeks to capitalize on the following:

I. It allows filtering of active vs. inactive records in queries.

II. It preserves historical data for analysis and auditing purposes.

III. It provides a safe alternative to deletion, preventing loss of important information.

IV. It allows for reactivation of courses, instructors, or learning outcomes if needed.

5) Normalization:

Response:
The proposed structure is in Third Normal Form (3NF). We've separated courses, learning outcomes, instructors, and course offerings into different tables to eliminate redundancy. In doing so, we ensure that the database can accommodate future changes to the program structure, course offerings, or instructional assignments without requiring structural modifications. The InstructorAssignments table serves as a mapping table to represent the many-to-many relationship between instructors and course offerings.

Explanation:
The motivation behind the current normalized structure is to minimize data redundancy and ensure data integrity. By adhering to the Third Normal Form (3NF), we've designed a database that is efficient, flexible, and less prone to anomalies.

The separation of entities into distinct tables (Courses, LearningOutcomes, Instructors, CourseOfferings, and InstructorAssignments) allows us to store each piece of information in one place. This approach offers the following three benefits:

I. By storing each piece of information once, we reduce the risk of inconsistencies that can arise from having the same data in multiple places (Data Consistency).

II. When information needs to change, it needs updating in one place, reducing the chance of partial updates leading to data discrepancies (Easy Updates).

III. The normalized structure allows for addition or modification of data without requiring changes to the schema (Flexible).

The InstructorAssignments mapping table resolves the many-to-many relationship between instructors and course offerings without introducing redundancy. This anticipates scenarios where multiple instructors teach a single course offering, or where one instructor teaches multiple course offerings, without duplicating instructor or course offering data.

6) Indexes:

Response:

- Primary keys: Automatically indexed
- Foreign keys: Add indexes to improve join performance
- Courses table: Index on IsActive
- Instructors table: Index on IsActive
- LearningOutcomes table: Index on IsActive and CourseID
- CourseOfferings table: Index on CourseID and Term
- InstructorAssignments table: Composite index on (OfferingID, InstructorID)

Explanation:
Our approach to indexing in this database design seeks to optimize query performance and ensure efficient record retrieval.

We leverage the auto-increment feature found in commonly used DBMS software for primary keys because it provides fast lookup times when we search for specific records or perform joins based on such keys. Adding indexes to foreign keys improves join performance. When tables are joined, the database can use these indexes to locate matching records more quickly. This reduces the time

required for complex queries involving multiple tables.

The IsActive field in the Courses, Instructors, and LearningOutcomes tables receives an index because of its anticipated frequent use when users filter for active and inactive records. This index allows for rapid retrieval of current courses, active instructors, or valid learning outcomes without evoking the increasingly cumbersome process of scanning entire tables.

In the LearningOutcomes table, a composite index on IsActive and CourseID supports queries that filter active learning outcomes for specific courses. We expect this combination to be a common query pattern when retrieving course information.

The CourseOfferings table includes an index on CourseID and Term. This index supports efficient searches for offerings of specific courses in particular terms. The composite index on OfferingID and InstructorID in the InstructorAssignments table helps queries that link course offerings with their assigned instructors.

7) Constraints to enforce:

Response:
FOREIGN KEY: Ensure referential integrity between tables
NOT NULL: On essential fields like course names, instructor names, etc.
CHECK:

- Ensure Term in CourseOfferings is 'Spring', 'Summer', or 'Fall'
- Ensure Year in CourseOfferings is valid (e.g., ≥ 2021)

UNIQUE:

- Distinct course codes in the Courses table
- Distinct combination of CourseID, Term, and Year in CourseOfferings

Explanation:
The implementation of constraints in our database design serves to maintain data integrity and prevent invalid data entry.

Foreign key constraints ensure referential integrity between tables. This prevents orphaned records and maintains the logical relationships between entities. For example, it prevents the assignment of a non-existent course to a course offering or an invalid instructor to a teaching assignment.

Not null constraints on necessary fields such as course names and instructor names guarantee that required data will always exist. It prevents incomplete

records that could lead to data inconsistencies or application errors.

Check constraints reflect the school's organizational logic within the database. The CourseOfferings.Term field limits values to 'Spring', 'Summer', or 'Fall', reflecting the academic calendar structure. The Year constraint ensures that only valid, future-dated offerings are entered, preventing historical data entry errors.

Unique constraints prevent duplicate entries where distinction is required. The unique course codes in the Courses table ensure that each course has a distinct identifier. The composite unique constraint on CourseID, Term, and Year in CourseOfferings prevents duplicate offerings of the same course in the same term and year.

The constraint-imposed database then has the following advantageous qualities:

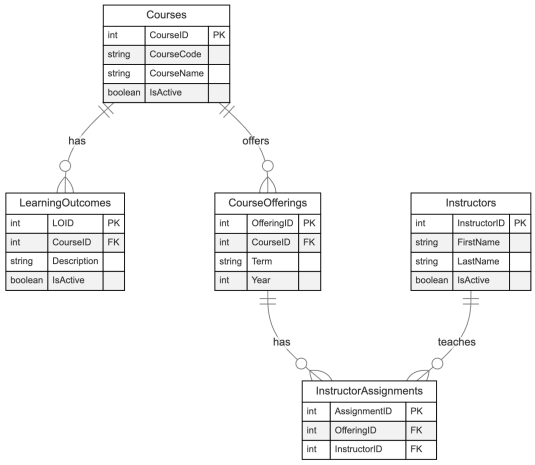
I. Logical consistency across related tables

II. Completeness of critical data points

III. Embedded enforcement of organizational logic in the db

IV. Avoidance of entity ambiguity arising from duplicate records

8) ER-Diagram



9) Extension to Residential Program:

Response:
a) Add new table:

- Programs(ProgramID, ProgramName, IsActive)

b) Modify existing tables:

- Courses: Add ProgramID (FK referencing Programs table)
- CourseOfferings: Add ProgramID (FK referencing Programs table)

c) New data needed:

- Program information (Online MSDS, Residential MSDS)
- Program-specific course information
- Learning outcome differences between programs

Explanation:
The extension of our database to support the Residential MSDS Program requires structural modifications and consideration of potential issues.

The addition of a Programs table allows for distinct representation of Online and Residential programs. This new entity becomes a central point of reference for program-specific data. Modifying the Courses and CourseOfferings tables with a ProgramID foreign key enables association of courses and offerings with specific programs. This modification supports scenarios where courses or offerings may differ between programs. New data requirements stem from the introduction of program-specific information. This includes basic program details, course variations between programs, and potential differences in learning outcomes.

Extension challenges include:

I. Shared courses between programs necessitate a decision between duplicate course entries or a many-to-many relationship between Courses and Programs.

II. Instructor assignments across both programs may require addition of ProgramID to the InstructorAssignments table for efficient querying.

III. Learning outcome variations between programs for the same course may necessitate modification of the LearningOutcomes table to include ProgramID.

IV. Potential differences in term structures between Online and Residential programs may require adjustments to the CourseOfferings table, such as adding ProgramID and modifying the Term field.

V. Capture of program-specific details like on-campus requirements, internships, or capstone projects may require additional tables or fields.