Information Technology Institute

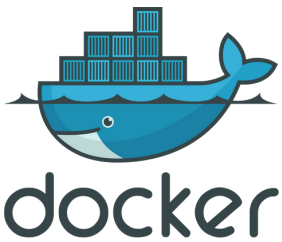
EXAMINATION SYSTEM

BY:

- AbdAelrahman Mostafa Mohamed
- Nora Magdy Mohamed Esmat
- Hussein Mohamed Suleiman
- Mina Essam Azmy
- Maher Mahmoud Elmoghazi

Project Overview

- **Purpose:**
  - To provide a secure, scalable, and well-structured database that manages the entire examination process. The system ensures accuracy, integrity, and efficiency in handling academic data, from exam creation to results calculation.
- **Scope:**
  - Covers user roles, courses, intakes, exams, question banks, answers, and results.
- **Objectives:**
  - Centralize all exam data
  - Fast, accurate result calculation
  - Support multiple intakes, branches, and courses
  - Scale easily as students and courses grow
- **Key Features**
  - Role-based access with multi-schema design
  - Flexible exam setup, auto/manual grading
  - Instant results
- **Security Considerations**
  - Role-based access control
  - Data integrity via constraints and authentication
- **Tech Stack**



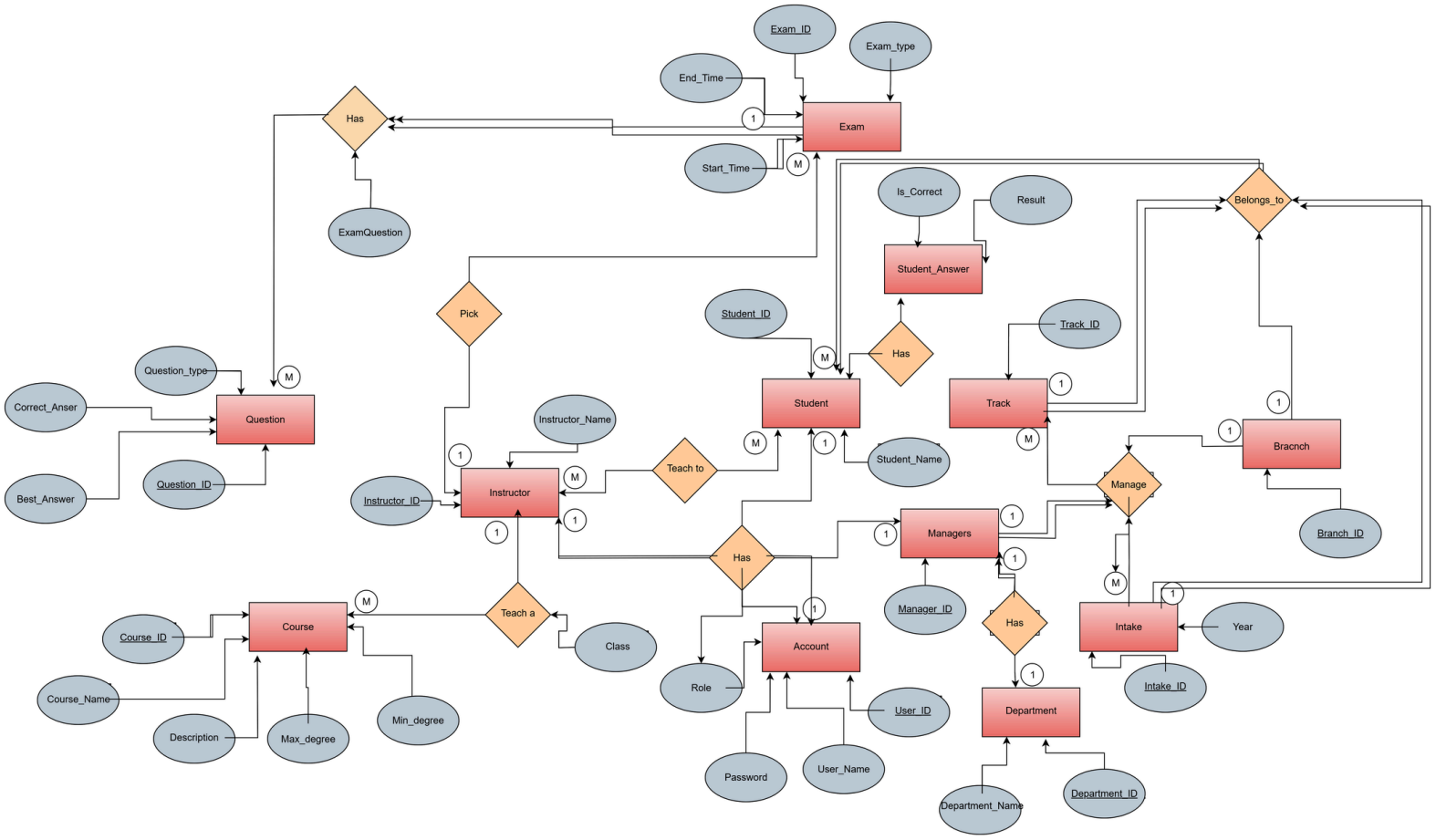
Conceptual Schema

THE EXAMINATION SYSTEM, OUTLINING ENTITIES, THEIR KEY ATTRIBUTES, AND THE MAIN RELATIONSHIPS BETWEEN THEM.

- **Entities:**
  - users (user_id, username, password, user_type, is_active, created_at, last_login)
  - managers (manager_id, user_id, name, email)
  - departments (department_id, name)
  - tracks (track_id, name, department_id)
  - branches (branch_id, name, location)
  - intakes (intake_id, intake_year)
  - instructors (instructor_id, user_id, name, email, department_id)
  - students (student_id, user_id, name, email, branch_id, track_id, intake_id)
  - courses (course_id, course_name, description, max_degree, min_degree)
  - class_offerings (class_id, course_id, instructor_id, intake_id, branch_id, track_id)
  - questions (question_id, course_id, question_text, question_type)
  - options (option_id, question_id, option_text, is_correct)
  - accepted_text_answers (text_answers_id, question_id, accepted_pattern)
  - exams (exam_id, exam_type, start_time, end_time, total_time, total_degree, class_id, extra_time_minutes, open_book, allow_calculator)
  - exam_questions (exam_id, question_id, question_degree)
  - exam_students (exam_student_id, exam_id, student_id, exam_date, start_time, end_time, total_grade, submission_time)
  - student_answers (student_answers_id, exam_student_id, question_id, option_id, answer_text, is_correct, manual_score, auto_match)
  - student_exam_results (exam_results_id, exam_student_id, exam_score)
  - class_results (class_results_id, student_id, class_id, total_score, status)

- **Relationships**
  - managers manages exactly one users account (1:1, total participation).
  - tracks belongs to one departments (M:1, total on track).
  - instructors are linked to one users account and belong to one departments (1:1 and M:1, total).
  - students have one users account and are assigned to one branches, one tracks, and one intakes (1:1 and M:1, total).
  - class_offerings are linked to one courses, one instructors, one intakes, one branches, and one tracks (M:1, total).
  - questions belong to one courses (M:1, total).
  - options and accepted_text_answers are linked to one questions (M:1, total).
  - exams are scheduled for one class_offerings (M:1, total).
  - exam_questions link exams with questions (M:N, total).
  - exam_students link students with exams (M:N, total).
  - student_answers link exam_students with questions and optionally options (M:N and M:1, partial on options).
  - student_exam_results are tied to one exam_students (1:1, total).
  - class_results summarize the performance of one students in one class_offerings (M:1, total).

Entity-Relationship Diagram (ERD)



ERD mapping organizing

This part explains how the designed ERD for the examination system is transformed into a fully functional SQL database.

- MAPPING GUIDELINES:
  - ENTITIES → TABLES:
    - EVERY ENTITY IN THE ERD IS REPRESENTED AS A DEDICATED TABLE WITHIN THE DATABASE, WITH PRIMARY KEYS CLEARLY DEFINED.
  - ATTRIBUTES → COLUMNS:
    - EACH ENTITY'S ATTRIBUTES ARE CONVERTED INTO TABLE COLUMNS, MAINTAINING THEIR ORIGINAL DATA TYPES AND CONSTRAINTS.
  - RELATIONSHIPS → FOREIGN KEYS / LINKING TABLES:
    - 1-TO-MANY (1:M): IMPLEMENTED BY ADDING A FOREIGN KEY IN THE TABLE ON THE "MANY" SIDE, REFERENCING THE "ONE" SIDE.
    - MANY-TO-MANY (M:N): IMPLEMENTED THROUGH INTERMEDIATE TABLES THAT STORE FOREIGN KEYS FROM BOTH RELATED TABLES, FORMING A COMPOSITE PRIMARY KEY.
    - SPECIALIZED ENTITIES: WHEN APPLICABLE, SUBTYPE TABLES INHERIT THEIR PRIMARY KEYS FROM THE MAIN ENTITY TABLE VIA FOREIGN KEY RELATIONSHIPS.
  - WEAK ENTITIES:
    - USE FOREIGN KEYS TO REFERENCE THEIR PARENT ENTITIES, WITH ADDITIONAL UNIQUE IDENTIFIERS IF REQUIRED FOR SYSTEM FUNCTIONALITY.
  - CONSTRAINTS:
    - INTEGRITY RULES FROM THE ERD (E.G., UNIQUENESS, ALLOWED VALUES, NON-NULL) ARE ENFORCED THROUGH SQL CONSTRAINTS LIKE UNIQUE, CHECK, AND NOT NULL.
  - NORMALIZATION & REDUNDANCY CONTROL:
    - ENSURE ALL TABLES FOLLOW NORMALIZATION RULES (UP TO AT LEAST 3NF) TO ELIMINATE REDUNDANCY AND MAINTAIN DATA INTEGRITY, UNLESS DENORMALIZATION IS INTENTIONALLY APPLIED FOR PERFORMANCE.
  - INDEXES & PERFORMANCE:
    - DEFINE INDEXES (SINGLE OR COMPOSITE) ON FREQUENTLY QUERIED COLUMNS AND FOREIGN KEYS TO OPTIMIZE QUERY PERFORMANCE WHILE BALANCING STORAGE AND WRITE COSTS.

Organizational Schemas

The examination system consists of multiple schemas, each containing entities that represent different aspects of the system.

- Core
  - core.users
  - core.managers
  - core.departments
  - core.tracks
  - core.branches
  - core.intakes
  - core.instructors
  - core.students
- Exams
  - exams.exams
  - exams.exam_questions
  - exams.exam_students
- Courses
  - courses.courses
  - courses.class_offerings
- questions_bank
  - questions_bank.questions
  - questions_bank.options
  - questions_bank.accepted_text_answers
- Answer
  - answers.student_answers
  - answers.student_exam_results
  - answers.class_results

For Security and organization we made 3 more schemas, one for each role manager - instructor - student

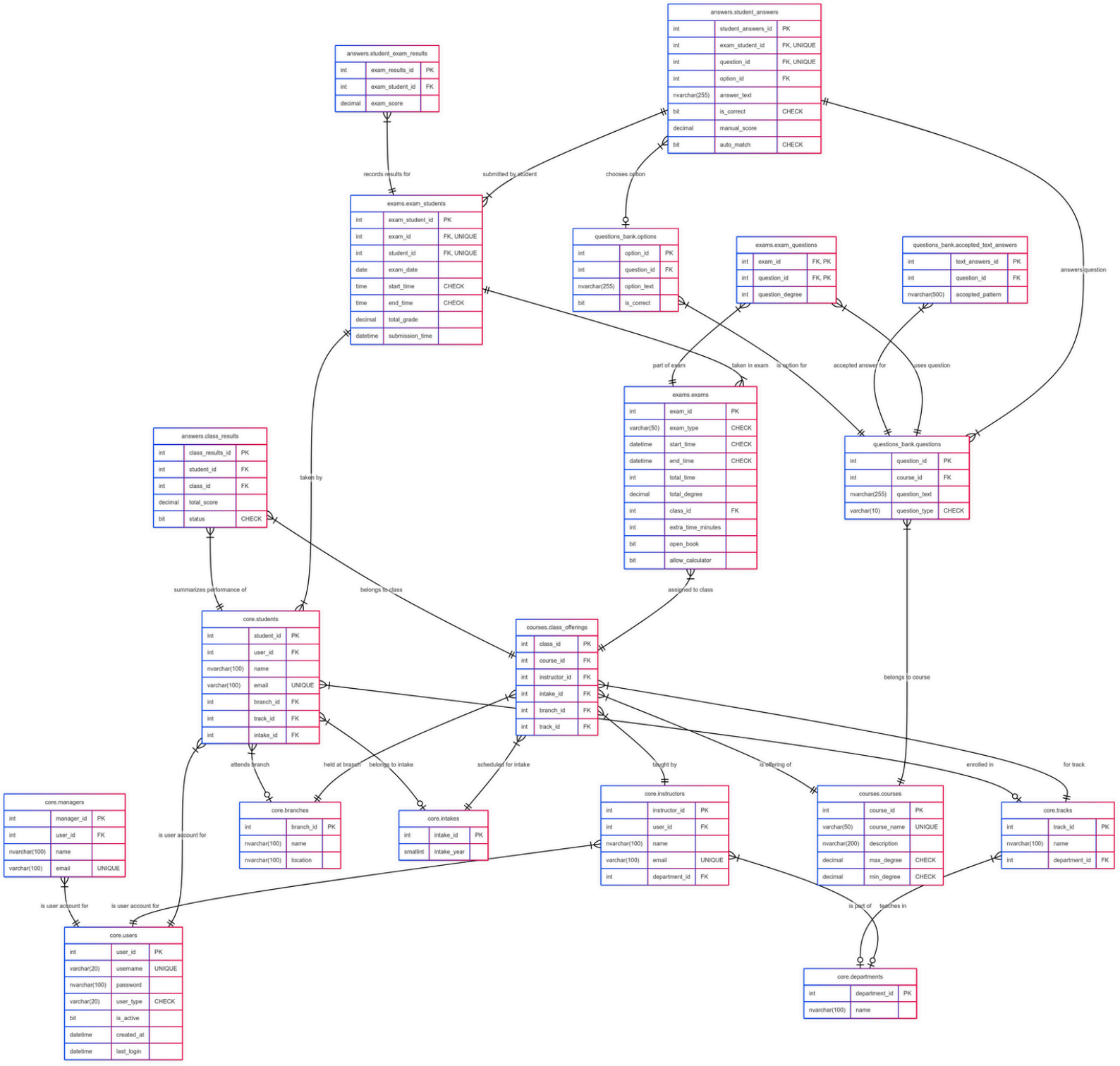
- manager_api
- instructor_api
- student_api

We prevented ant type of users to select, insert, update or delete on the main schemas(core, courses,....)
Every type of users can only deal with his own schema

```
-- Server role creation(manager)
-- give this role the permisions to alter logins
use master;
create server role manager;
grant alter any login to manager;
create login Osama with password = 'P@ssw0rd';
alter server role manager add member Osama;

use g1_examination_system;
create user Osama for login Osama;
create role db_manager;
alter role db_manager add member Osama

grant alter any user to db_manager;
grant alter any role to db_manager;

deny select, insert, update, delete on schema::core to db_manager;
deny select, insert, update, delete on schema::courses to db_manager;
deny select, insert, update, delete on schema::questions_bank to db_manager;
deny select, insert, update, delete on schema::exams to db_manager;
deny select, insert, update, delete on schema::answers to db_manager;
deny select, insert, update, delete on schema::dbo to db_manager;
deny execute on schema::instructor_api to db_manager;
deny select on schema::instructor_api to db_manager;
deny execute on schema::student_api to db_manager;
deny select on schema::student_api to db_manager;

grant execute on schema::manager_api to db_manager;
grant select on schema::manager_api to db_manager;
```

ERD after mapping

This diagram, created using Mermaid, visually represents the entities, attributes, and relationships of the examination system database. It illustrates how data components interact, including user management, courses, exams, questions, answers, and results



SQL DDL Implementation

Schema: core

The core schema contains the fundamental entities of the examination system, including users, managers, departments, tracks, branches, intakes, instructors, and students, along with their relationships.

```
use g1_examination_system
go
create schema core
go

create table core.users(
    user_id int primary key identity(1,1),
    username varchar(20) unique not null,
    password nvarchar(100) not null,
    user_type varchar(20) not null,
    is_active bit,
    created_at datetime not null default getdate(),
    last_login datetime null,

    constraint CK_users_usertype
        check (user_type in ('student', 'instructor', 'manager'))
) on fg_core
go

create table core.managers(
    manager_id int primary key identity(1,1),
    user_id int not null,
    name nvarchar(100),
    email varchar(100) unique,

    constraint FK_manager_users foreign key(user_id)
        references core.users(user_id)
) on fg_core
go

create table core.departments(
    department_id int Primary key identity(1, 1),
    name nvarchar(100)
) on fg_core
go

create table core.tracks(
    track_id int Primary key identity(1, 1),
    name nvarchar(100),
    department_id int,

    constraint FK_tracks_departments foreign key(department_id)
        references core.departments(department_id)
) on fg_core
go

create table core.branches(
    branch_id int Primary key identity(1, 1),
    name nvarchar(100),
    location nvarchar(100)
) on fg_core
go

create table core.intakes(
    intake_id int primary key identity(1, 1),
    intake_year smallint
) on fg_core
go
```

```
create table core.instructors(
    instructor_id int primary key identity(1, 1),
    user_id int not null,
    name nvarchar(100),
    email varchar(100) unique,
    department_id int,

    constraint FK_instructors_users foreign key(user_id)
        references core.users(user_id),
    constraint FK_instructors_departments foreign key(department_id)
        references core.departments(department_id)

) on fg_core
go

create table core.students(
    student_id int primary key identity(1, 1),
    user_id int not null,
    name nvarchar(100),
    email varchar(100) unique,
    branch_id int,
    track_id int,
    intake_id int,

    constraint FK_students_users foreign key(user_id)
        references core.users(user_id),
    constraint FK_students_branches foreign key(branch_id)
        references core.branches(branch_id),
    constraint FK_students_tracks foreign key(track_id)
        references core.tracks(track_id),
    constraint FK_students_intakes foreign key(intake_id)
        references core.intakes(intake_id)
) on fg_core
go
```

SQL DDL Implementation

Schema: courses

The courses schema manages course details and class offerings, defining course information, grading thresholds, and the assignment of courses to instructors, intakes, branches, and tracks.

```
use g1_examination_system
go
create schema courses
go

create table courses.courses(
    course_id int primary key identity(1, 1),
    course_name varchar(50) not null,
    description nvarchar(200),
    max_degree decimal(5, 2) not null default 100.00,
    min_degree decimal(5, 2) not null default 60.00,

    constraint uq_course_name unique(course_name),
    constraint CK_courses_degree_range
        check(
        max_degree > min_degree
        and max_degree <= 100.00
        and min_degree >= 60.00
        )
) on fg_courses
go

create table courses.class_offerings(
    class_id int primary key identity(1, 1),
    course_id int not null,
    instructor_id int not null,
    intake_id int not null,
    branch_id int not null,
    track_id int not null,

    constraint FK_class_instructor foreign key(instructor_id)
        references core.instructors(instructor_id),
    constraint FK_class_intakes foreign key(intake_id)
        references core.intakes(intake_id),
    constraint FK_class_branches foreign key(branch_id)
        references core.branches(branch_id),
    constraint FK_class_tracks foreign key(track_id)
        references core.tracks(track_id),
    constraint FK_class_courses foreign key(course_id)
        references courses.courses(course_id)
) on fg_courses
go
```

SQL DDL Implementation

Schema: questions_bank

The questions_bank schema stores all exam questions, including multiple-choice, true/false, and text-based questions, along with their possible options and accepted answer patterns.

```
use g1_examination_system
go
create schema questions_bank
go

create table questions_bank.questions(
    question_id int not null primary key identity(1, 1),
    course_id int not null,
    question_text nvarchar(255) not null,
    question_type varchar(10) not null,

    constraint CK_question_question_type
        check (question_type in ('MC', 'TF', 'text')),
    constraint FK_questions_courses foreign key(course_id)
        references courses.courses(course_id)
) on fg_questions
go

create table questions_bank.options (
    option_id int primary key identity(1, 1),
    question_id int not null,
    option_text nvarchar(255) not null,
    is_correct bit not null default 0,

    constraint FK_options_questions foreign key(question_id)
        references questions_bank.questions(question_id)
) on fg_questions
go

create table questions_bank.accepted_text_answers (
    text_answers_id int primary key identity(1, 1),
    question_id int not null,
    accepted_pattern nvarchar(500) not null,

    constraint FK_textanswers_questions foreign key(question_id)
        references questions_bank.questions(question_id)
) on fg_questions
go
```

SQL DDL Implementation

Schema: exams

The exams schema defines exam details, assigned questions, and student participation records, including scheduling, constraints, and links to related courses, classes, and students.

```
use g1_examination_system
go
create schema exams
go

create table exams.exams (
    exam_id int primary key identity(1, 1),
    exam_type varchar(50) not null default 'main',
    start_time datetime not null,
    end_time datetime not null,
    total_time int not null,
    total_degree decimal(5, 2) not null,
    class_id int not null,
    extra_time_minutes int null default 0,
    open_book bit not null default 0,
    allow_calculator bit not null default 0,

    constraint ck_exam_type
        check(exam_type in ('main', 'corrective')),
    constraint fk_exams_class foreign key(class_id)
        references courses.class_offerings(class_id),
    constraint ck_exams_time check (end_time > start_time)
) on fg_exams
go

create table exams.exam_questions (
    exam_id int not null,
    question_id int not null,
    question_degree int not null,

    constraint pk_exam_questions primary key (exam_id, question_id),
    constraint fk_exam_questions_exam foreign key(exam_id)
        references exams.exams(exam_id),
    constraint fk_exam_questions_question foreign key(question_id)
        references questions_bank.questions(question_id)
) on fg_exams
go

create table exams.exam_students (
    exam_student_id int primary key identity(1, 1),
    exam_id int not null,
    student_id int not null,
    exam_date date not null,
    start_time time not null,
    end_time time not null,
    -- drop the total_grade
    submission_time datetime null,

    constraint fk_exam_students_exam foreign key(exam_id)
        references exams.exams(exam_id),
    constraint fk_exam_students_student foreign key(student_id)
        references core.students(student_id),
    constraint ck_exam_students_time check (end_time > start_time),
    constraint uq_exam_students unique (exam_id, student_id)
) on fg_exams
go
```

SQL DDL Implementation

Schema: answers

The answers schema stores students' submitted answers, individual exam results, and aggregated class results, enabling accurate grading and performance tracking.

```
use g1_examination_system
go
create schema answers
go

create table answers.student_answers(
    student_answers_id int primary key identity(1,1),
    exam_student_id int not null,
    question_id int not null,
    option_id int null,
    answer_text nvarchar(255),
    is_correct bit default 0,
    manual_score decimal(5, 2) null,
    auto_match bit default 0,

    constraint fk_studentanswers_examstudent foreign key(exam_student_id)
        references exams.exam_students(exam_student_id),
    constraint fk_studentanswers_question foreign key(question_id)
        references questions_bank.questions(question_id),
    constraint fk_studentanswers_option foreign key(option_id)
        references questions_bank.options(option_id),
    constraint uq_questions_examstudent unique(exam_student_id, question_id),
    constraint ck_auto_match check(auto_match in (0, 1)),
    constraint ck_is_correct check(is_correct in (0, 1))
) on fg_answers
go


create table answers.student_exam_results(
    exam_results_id int primary key identity(1, 1),
    exam_student_id int not null,
    exam_score decimal(5,2),

    constraint fk_exresults_student foreign key(exam_student_id)
        references exams.exam_students(exam_student_id)
) on fg_answers
go

create table answers.class_results(
    class_results_id int primary key identity(1, 1),
    student_id int not null,
    class_id int not null,
    total_score decimal(5,2),
    status bit not null default 0,

    constraint fk_classresults_student foreign key(student_id)
        references core.students(student_id),
    constraint fk_classresults_class foreign key(class_id)
        references courses.class_offerings(class_id),
    constraint ck_status check(status in (0, 1))

) on fg_answers
go
```

SQL Seed Data

Insertion examples show how initial data is populated into the database tables

```
use g1_examination_system;
go

insert into core.users(username, password, user_type, is_active)
values
('Osama', '123pass', 'manager', 1),
('Sarah', '123pass', 'instructor', 1),
('Youmna', '123pass', 'instructor', 1),
('Hussein7', '123pass', 'student', 1),
('abdlrhman12', '123pass', 'student', 1),
('Mina44', '123pass', 'student', 1),
('Nora23', '123pass', 'student', 1),
('Maher88', '123pass', 'student', 1)
go

insert into courses.courses(course_name, description, max_degree, min_degree)
values
('sql fundamentals', 'introduction to sql and database management', 100.00, 60.00),
('python programming', 'basic to intermediate python programming', 100.00, 60.00),
('data analysis', 'data analysis using python and pandas', 100.00, 60.00)
go

insert into questions_bank.questions(course_id, question_text, question_type)
values
(1, 'what does sql stand for?', 'mc'),
(1, 'select statement is used to retrieve data from database', 'tf'),
(1, 'write the sql command to create a table named students', 'text'),
(1, 'which clause is used to filter rows in sql?', 'mc'),
(1, 'primary key can contain null values', 'tf'),

(2, 'which of the following is the correct way to create a list in python?', 'mc'),
(2, 'python is case-sensitive language', 'tf'),
(2, 'write a python function to calculate factorial of a number', 'text'),
(2, 'what is the output of: print(type(5.0))?', 'mc'),
(2, 'python uses indentation to define code blocks', 'tf'),

(3, 'which library is commonly used for data manipulation in python?', 'mc'),
(3, 'pandas dataframe can store data of different types', 'tf'),
(3, 'write code to read a csv file using pandas', 'text'),
(3, 'what method is used to get basic statistics of a dataframe?', 'mc'),
(3, 'missing values in pandas are represented by nan', 'tf')
go

insert into questions_bank.options(question_id, option_text, is_correct)
values
(1, 'structured query language', 1),
(1, 'simple query language', 0),
(1, 'standard query language', 0),
(1, 'system query language', 0),

(4, 'where', 1),
(4, 'select', 0),
(4, 'from', 0),
(4, 'order by', 0),

(6, '[1, 2, 3, 4]', 1),
(6, '(1, 2, 3, 4)', 0),
(6, '{1, 2, 3, 4}', 0),
(6, 'list(1, 2, 3, 4)', 0),
```

Database Objects

Object Name	Type	Description
manager_api. sp_add_user	SP	Creates a new user with database records and SQL Server authentication. Automatically assigns user to appropriate tables and roles based on user type.
manager_api. sp_update_user_auth	SP	Updates user information including credentials, roles Handles SQL Server login changes.
manager_api. sp_update_user_details	SP	Updates user personal details on each core.student, instructor, manager based on the type.
manager_api. sp_delete_user	SP	Safely removes users from database and SQL Server authentication. Ensures proper cleanup sequence to maintain data integrity.
manager_api. sp_edit_track	SP	Creates a new track under a department or updates an existing track's information.
manager_api. sp_edit_branch	SP	Creates a new branch if @BranchId is null, otherwise updates an existing branch's name and location.
manager_api. sp_edit_intake	SP	Adds a new intake year or updates the year for an existing intake.
manager_api. sp_edit_department	SP	Creates a new department or updates an department's information.
manager_api. sp_delete_core_entity	SP	Deleting a record from one of the core tables: branch, track, intake, or department. based on the type of entity and its ID.

Database Objects

Object Name	Type	Description
manager_api. vw_all_users	View	Comprehensive view that consolidates all user information across different user types (managers, instructors, students) with their respective details, contact information, and organizational associations.
manager_api. vw_system_dashboard	View	High-level system overview providing key metrics including active/inactive users count, total counts for each user type, courses, classes, exams, and questions.
manager_api. vw_course_performance	View	Detailed course analytics showing performance metrics including enrollment statistics, exam scores, pass/fail rates, and overall course effectiveness indicators.
manager_api. vw_instructor_performance	View	Instructor evaluation metrics displaying teaching load, student outcomes, average scores, and success rates for performance assessment and review.
manager_api. vw_student_progress	View	Individual student tracking view showing enrollment history, academic performance, class completion status, and progress indicators across their academic journey.
manager_api. vw_exam_details	View	Comprehensive exam information including scheduling details, participation statistics, submission rates, and performance analytics for exam administration and monitoring.
manager_api. vw_questions_overview	View	Question bank analytics providing insights into question distribution by type, course coverage, and success rates for assessment quality evaluation.
manager_api. vw_departments_overview	View	Department-level statistics showing instructor count, track offerings, and student enrollment numbers for organizational planning and resource allocation.
manager_api. vw_branches_overview	View	Branch performance metrics including student enrollment, class offerings, course availability, and location-based statistics for multi-branch management.

Database Objects

Object Name	Type	Description
instructor_api.sp_manual_score_update	SP	updates the manual score for a specific student answer, then recalculates the student's final exam result to reflect the change.
instructor_api.sp_compute_final_exam_result	SP	recalculates a student's total exam score based on their answers, determines pass/fail status, and updates or inserts the corresponding records in both student_exam_results and class_results.
instructor_api.sp_search_exam	SP	Searches for exams by optional filters such as course name, exam type, instructor, or exam date, returning matching exam and related course/instructor details.
instructor_api.sp_search_students	SP	Searches for students using optional filters such as name, email, or intake ID, returning their basic details.
instructor_api.sp_add_exam	SP	Adds a new exam for a specific course and instructor, validates teaching assignment and total degree limits, then inserts exam details and its questions into the database.
instructor_api.fn_check_text_answers	Function	Validates a text answer by normalizing it (case-insensitive, punctuation removed) and checking if it matches any accepted answer patterns for the given question(Bonus).
instructor_api.fn_check_mcq_answers	Function	Checks if a given multiple-choice option is correct and returns 1 for correct or 0 for incorrect.

Database Objects

Object Name	Type	Description
instructor_api.vw_questions	View	Displays all questions from the question bank for the courses that instructors are assigned to teach, including course and instructor details.
instructor_api.vw_question_types	View	Lists the distinct question types available in the question bank (e.g., multiple choice, true/false, essay)
instructor_api.vw_question_options	View	Shows options for multiple-choice and true/false questions, including whether each option is correct.
instructor_api.vw_accepted_text_answers	View	Contains accepted answer patterns for text-based questions to support automated grading.
instructor_api.vw_courses	View	Displays details of courses assigned to each instructor, including course name, description, and grading criteria.
instructor_api.vw_class_offerings	View	Lists class offerings for each course, including branch, track, and intake information.
instructor_api.vw_exams	View	Shows exam details for courses assigned to instructors, including schedule, duration, and total marks.
instructor_api.vw_exam_question_details	View	Displays questions linked to each exam along with their assigned marks and related instructor/course info.
instructor_api.vw_exam_students	View	Lists students assigned to exams, with exam schedule, permissions, grades, and submission time.
instructor_api.vw_student_result	View + CTE	Shows a student's final results across exams or courses, combining scores and status.
instructor_api.vw_exam_details	View + CTE	Provides full exam details, including exam info, instructor, course, and related questions.

Database Objects

Object Name	Type	Description
student_api. sp_get_dashboard	SP	Returns the student's profile information along with statistics about total and incomplete exams.
student_api. sp_get_student_exams	SP	Returns the list of upcoming exams for the student with course and exam details.
student_api. sp_take_exam	SP	Handles student answer submission by validating eligibility, checking correctness for objective questions, assigning scores when applicable, and inserting the answer into student_answers.
student_api. sp_submit_answer	SP	Retrieves all questions (and options if applicable) for a given exam, ensuring the student is authorized to take it.
student_api.tr_start_exam	Trigger	Prevents a student from starting an exam outside its scheduled start time by validating the current time when a record is inserted into exam_students.
student_api.tr_exam_end_time	Trigger	Ensures a student cannot continue an exam once its scheduled end time is reached by checking the current time when a record is inserted into exam_students.
student_api.trg_auto_grade_student_answer	Trigger	Automatically grades new or updated student answers, assigns scores where applicable, and recalculates the related exam and class results.

Database Objects

Object Name	Type	Description
ix_students_user_id	Index	Speeds up lookups and joins between students and user accounts.
ix_students_track_intake_branch	Index	Optimizes reporting on students by track, intake, and branch.
ix_class_offerings_intake_id	Index	Speeds up queries filtering classes by intake.
ix_options_question_id	Index	Improves performance when retrieving options for a given question.
ix_exam_students_exam_id	Index	Speeds up listing students assigned to a specific exam.
ix_student_answers_question_id	Index	Optimizes retrieval of student answers for a specific question.
ix_student_answers_examstudent_question	Index	Improves exam correction queries by quickly locating a student's answer to a given question.

Daily Full Backup

Automated Database Backup Job
This script uses SQL Server Agent's built-in stored procedures to create a scheduled job for performing daily backups of the g1_examination_system database
The job is configured in the msdb system database and consists of a backup step, a daily schedule, and an association with the SQL Server instance. The backup file is saved with a timestamp in the specified directory, ensuring regular and consistent data protection.

first: we create the /var/opt/mssql/backups/ directory in the container
second: we make sure sql agent is running, run the command inside the docker
/opt/mssql/bin/mssql-conf set sqlagent.enabled true

```
use msdb;
go

exec sp_add_job @job_name = N'dailydbbackup'
go

exec sp_add_jobstep
    @job_name = N'dailydbbackup',
    @step_name = N'backupstep',
    @subsystem = N'TSQL',
    @command = N'
        BACKUP DATABASE g1_examination_system
        TO DISK = ''/var/opt/mssql/backups/g1_examination_system.bak''
        WITH INIT, COMPRESSION;
    '
-- WITH INIT: overwrites existing backup file
-- WITH COMPRESSION: reduces backup file size and speeds up backup

go

exec sp_add_schedule
    @schedule_name = N'dailybackupschedule',
    @freq_type = 4,
    @freq_interval = 1,
    @active_start_time = 020000
go

exec sp_attach_schedule
    @job_name = N'dailydbbackup',
    @schedule_name = N'dailybackupschedule'
go

exec sp_add_jobserver @job_name = N'dailydbbackup'
go
```

Daily Full Backup

Another method using crontab inside of the container

- Enter as root
  - docker exec -it --user root mssql-exam-system-iti /bin/bash
- Install vim
  - apt-get update
  - apt-get install vim -y
- Write the code
  - mkdir /var/opt/mssql/backups/
  - vim /var/opt/mssql/backups/backup.sql

```
BACKUP DATABASE g1_examination_system
TO DISK = '/var/opt/mssql/backups/g1_examination_system.bak'
WITH INIT, COMPRESSION;
GO
```

- Make the sh backup code
  - chown mssql:mssql /var/opt/mssql/backups/
  - chmod a+w /var/opt/mssql/backups/
  - ln -s /opt/mssql-tools18/bin/sqlcmd /usr/local/bin/sqlcmd
  - vim /var/opt/mssql/backups/run_backup.sh

```
#!/bin/bash
sqlcmd -S localhost -U sa -P 'P@ssw0rd' -i /var/opt/mssql/backups/backup.sql -C
```

- Install cron
  - apt-get install cron -y
  - service cron start
  - crontab -e
  - insert

```
00 22 * * * /bin/sh /var/opt/mssql/backups/run_backup.sh >>
/var/opt/mssql/backups/backup.log 2>&1
```