DBMS Documentation for Assignment: Part A

Requirements Specification

|  |  |
| --- | --- |
| **Entity** | **Documentation** |
| Admin Office | * The admin office must be granted access to most of the data since they must handle most of the information. * Access will be implemented in the programmable part with the GRAND Command. |
| Academic Periods | * Each module must have an academic period in order to be unique within the year. * The module must have a partial key coming from the Academic Period entity so that two modules can not have the same key throughout the years. |
| Modules | * Modules must be separated into two entities. * One entity (Modules) must include all the modules of a programme. * The other entity (Module\_Per\_Year) are the active modules that belong to an academic period. |
| Sessions | * Sessions must be separated into two entities * One entity (Session) includes all the standard sessions throughout the year. * The other entity (Lecture) is the actual session that is taught by a Lecturer. * A second entity was created in case details from the session have changes once. * If a change happens once, it does not mean that the change is permanent. Consequently, a different entity must be created in order to make changes. |
| Enrollment | * Enrollment is a procedure that can be implemented in the programmable part of the assignment. * No need for an entity. |
| Reports | * Procedure that must be done by the Admin office. * Admin office must have access to most data. * No need for an entity since this can be implemented in the programmable part. |
| Student Support Office | * Must have access to Student Attendance Records. * Code must be included in the programmable part in order to calculate if a student is being at risk. |
| Academics | * Person that must have access to their timetable and that of their modules. * Additionally, they must have access to attendance and timetable of the students that are enrolled on their modules. * Must belong to a Department unless they are Head of School or Rector. |
| Students | * Person that must have access to their timetable and attendance records. * Must belong to a programme. |
|  |  |
| University | Everything belongs to the university. |
| Schools | * For example:  1. **School of Sciences**  * Every University has Schools which differentiate each course |
| Departments | * For example:  1. **Computer Science** 2. **Mathematics**  * Every School must have different Departments |
| Programmes | * For example:  1. **BSc (Hons) Computing 4/Y** 2. **MSc Computing**  * Programmes are a collection of modules that should be taken by a student * Students belong to Programmes |

Additional Details:

* Every person must own a user account. For each user, the system must verify via username/password for before accessing the application.
* Students must own a student library card with a unique number so that they can scan their card to record their attendance.

Conceptual Design

|  |  |  |
| --- | --- | --- |
| **Entity** | **Documentation** | **Details** |
| PERSON | * Person is the top hierarchy of all people. * It separates into Academics and Students (disjoint). * Every person must own an account. | Every person must belong to the University |
| ACCOUNT | * A person must own only one account * One account must be managed only by one person | One-to-One relationship with entity PERSON. Decisions must be made for the relational model. |
| ACADEMICS | * Academics are separated into Office Academics, Rectors, and Academic Staff (Lecturers). * The reason to that is because Rectors do not belong to departments, but they are still academics. Plus, they all have different relationships with other entities. |  |
| OFFICE ACADEMICS | * Office Academics separate into Admin Office Academics and Student Support Office Academics. * No functionalities. | Their function can be done in the programmable part |
| RECTOR | * A Rector must manage a University. * A Rector must manage Schools. | Different persons manage University and Schools |
| UNIVERSITY | * All Persons belong to the University * University has different type of Schools | A University must have a Rector |
| SCHOOL | * All different Schools belong to a University * Each type of School has different Departments | All Schools must have a Rector |
| DEPARTMENT | * Each Department has different Modules * Each Department has Programmes |  |
| PROGRAMMES | * Every Programme includes Module\_Per\_Year which includes all the modules of the year that are active and taken by Students | Students belong to Programmes. |
| MODULE\_ PER\_YEAR | * Each active Module must have an Academic Period so that they are unique. * Ex. CO2701: Database Systems 2003 is not the same as CO2701: Database Systems 2020. | Not all Modules are active every semester |
| ACADEMIC PERIOD | * Every active Module must belong to an Academic Period as a partial Key. |  |
| MODULES | * Entity Modules includes all active or in-active Modules for every Department. |  |
| SESSION | * All standard Sessions throughout the year are in this entity including all the details for each Session. * Sometimes some things might change in the Session.   Ex. Lecturer, time or location. | Changes might occur in a session that may only happen once. |
| LECTURE | * Lecture is the actual Lecture that occurs with changes or not changes. * Each Lecture can be taught by one or more Lecturers. | Lecture includes very similar data with Session |
| ACADEMIC\_ STAFF | * Academic Staff are the Lecturers that teach classes. * One Academic Staff is appointed ass the Department Manager. * All Academic Staff must belong to a Department | Grades are generated through the relationship Lecture-Academic |
| STUDENTS | * Zero one many Students may attend a Lecture * Students belong to Programmes | Students must punch their card for their attendance |

A close up of a map

Description automatically generated

Logical Design

|  |  |  |
| --- | --- | --- |
| **Relationship** | **Type** | **Documentation** |
| Person - Account | 1:1 | * Approach:   Foreign Key.   * Key of Account is a foreign key in Person. |
| University - Rector | 1:1 | * Approach:   Foreign Key.   * Key of Rector is a foreign key in University. |
| School - Rector | 1:1 | * Approach:   Foreign Key.   * Key of Rector is a foreign key in School. |
| Department – Academic Staff | 1:1 | * Approach:   Foreign Key   * Key of Academic Staff is a foreign key in Department. |
| Lecture – Student | M:N | * New table named:   Attendance.   * Includes the attendance of Students. * New table Attendance includes the keys of both Lecture and Student, along with a new column with Boolean value for Attendance Y/N. |
| Academic Staff – Lecture | M:N | * New table named:   Teach   * Includes the grades of students * New table Teach includes the keys of both Academic Staff and the Lecture along with a new column named Grades which will be calculated in the programmable part. |

Hierarchy Documentation:

A close up of a map

Description automatically generated

* All hierarchies are disjoint because they can not be overlapped. For instance, a student cannot be academic staff nor the opposite.
* For the implementation of the relational model, as far as the hierarchy is concerned, I used the 8B approach since all subclasses have total participation. Additionally, it is less redundant, and information can be retrieved fast since it is without joins of different tables. Furthermore, it eliminates all nulls which is a big advantage.
* The reasons for not choosing the other options are:
  1. 8A: There would be too confusing and redundant to retrieve information in such format since we need every subclass to join the superclass.
  2. 8C: 8C is fast and a big advantage of it is that it can support both partial and total participation. However, this option has many nulls for attributes.
  3. 8D: Like 8C, it has too many nulls that lead to redundancy.