School Tutoring Center

Kevin Velado

Database Systems

Table of Contents

Fact Finding and Information Gathering

Techniques Used

In order to better facilitate the needs of a company, fact finding is required in order to know what their database will be used for. Fact finding will better help me in knowing what the business will need to keep track of and further operations on that data. Two techniques were the most useful in the area of fact finding, those being Interviewing and Data Analysis.

Interviewing the employee's allowed me to create the necessary entity classes for the database. Meetings with the tutors were more or less informal. The main discussion with the tutors focused on keeping track of data and how their current system is not very sophisticated but works for what they need. The discussion with the director of the tutoring center was more formal and focused on what they want to do with the data. Essentially the later interview gave me insight on the operations that they wanted to do with the data. At the end of the interview process I was able to create the entity types required for the database and their most basic relationship. Although details were gathered about each type, I needed more information. So the next fact finding method filled in those blanks.

The next step in the fact finding process was data analysis. The tutoring center was using a very simple method to keep track of how many students came in for tutoring each day and what they needed help in. Essentially they created an excel spreadsheet and asked for the students to fill out a row. Looking at the data I was able to see what additional details each entity type needed.

Introduction to Enterprise/Company

The business I am focusing on for this project can be considered small. The Tutoring Center is based on campus and is comprised of a small number of employees. The Tutoring Center's focus is on the student's understanding of a subject. The center has tutors for the primary subjects that all students that attend the university will take. The two most important subjects are Mathematics and English, These two subjects see the most traffic flow and have the most tutors allocated to. With that being said all of the Tutoring Center is important, especially for the development of the students.

Structure of the Enterprise

The Tutoring Center is comprised of four main sections. They have a dedicated Computer Science and Mathematic area, meaning only one subject is tutored in one location. The reason being is that for Computer Science there is no other location on campus that facilitate the needs for computer science tutoring and for mathematics the traffic flow of students needing help is so great that they were granted their own tutoring location. Then there are two more locations that are for the purposes of tutoring English and other language based subjects and Biology, Physics and the other sciences that have not been listed. How the center works is as follows, during operating hours there are somewhere between 1 and 4 tutors working in one of the locations. Students will come in and sign in, meaning enter the data that the tutoring center wants (name, date, subject, etc.). Then they will proceed to sit down somewhere in the tutoring center and enlist the help of the tutors when they require it. After the

students are done, they simply sign out and leave to come another day. Since the Tutoring Center is being funded by a grant on the campus, they do not have to worry about a staff or payment to the tutors themselves. Also each tutoring center will have a head tutor, who will manage the tutors and be in charge of the data being correct and making any necessary changes.

Itemized Description of Major Objects

The Tutor entity is one of the two main entities of the system. The purpose of this entity is to represent the tutors in the relationships of the database system. The tutor entity will have relations with two other entities: Timeblock and Student. The Timeblock entity is a lesser entity and is used mainly to define at what hours the tutors will work. Whilst Student is the other main Entity type of the database. Student is quite possibly the most important entity due to the fact that the reason a database is being created is for the sole purpose of keeping track of the number of students coming in the tutoring center. The Student entity then in turn has a relationship with a Class entity and a Department entity. Class will be what the class the student is coming in for tutoring and the students major will be determined by the department.

Data Views and Operations for User Groups

The head tutor will be in charge of each tutoring facility and determine tutor grades and changes to time blocks if necessary. The head tutors only have this privilege in their respected section of the tutoring center. Meaning that the mathematics's head tutor can only change the tutor grades and time blocks of the tutors in the mathematics tutoring center. All the tutors will have no access to the database itself. Operations on the data will be made by the director of the Tutoring Center.

Conceptual Database Design

Entity Set Description

Tutor

- This entity type describes the tutors working in the tutoring center. This entity is mainly created to keep track of the tutors and how well they are doing.
- Candidate Key(s): TutorID
- Primary Key: TutorID
- Strong/Weak Entity: Strong
- Fields to be Indexed: TutorID, Name
- Attributes:

Name	**TutorID**	**Name**	**TutorGrade**	**PhoneNumber**	**PayRate**
Description	Tutor ID Number	The Name of the Tutor	A tutoring grade for the tutor	The phone number of the tutor	Determines the pay rate for the tutor
Domain/Type	Unsigned Integer	String	Character	Integer	Double
Value Range	0...2^32	Any	A,B,C,D,F	(000)000-0000 - (999)999-9999	00.00 – 99.99
Default Value	None	None	None	None	10.5
Nullable	No	No	No	Yes	No
Unique	Yes	No	No	Yes	No
Single or Multiple value	Single	Single	Single	Single	Single
Simple or Composite	Simple	Composite	Simple	Composite	Simple

TimeBlock

- This entity type comprises of the shifts that the tutoring center is open for. For example Shift 1 will be from 9:30 am to 10:45 am.
- Candidate Key: TimeID
- Primary Key: TimeID
- Strong/Weak Entity: Strong
- Fields to be Indexed: TimeID
- Attributes:

Name	**TimeID**	**StartTime**	**EndTime**
Description	The Time slot's ID	Beginning time of the shift	Ending time of the shift
Domain/Type	Unsigned Integer	Unsigned Integer	Unsigned Integer
Value Range	0...2^32	09:30 – 22:00	09:30 – 22:00
Default Value	None	None	None
Nullable	No	No	No
Unique	Yes	No	No
Single or Multiple value	Single	Single	Single
Simple or Composite	Simple	Composite	Composite

Subject

- This entity class contains the subjects that the tutoring center tutors in.
- Candidate Key: SubjectID
- Primary Key: SubjectID
- Strong/Weak Entitiy: Strong
- Fields to be Indexed: SubjectID
- Attributes:

Name	**SubjectID**	**SubjectDescription**
Description	The ID of the subject	A description of the subject
Domain/Type	String	String
Value Range	Any	Any
Default Value	None	None
Nullable	No	Yes
Unique	Yes	Yes
Single or Multiple value	Single	Single
Simple or Composite	Simple	Simple

Student

- This entity is to represent the students that come to the tutoring center for tutoring. The data that comes from this entity will be used to better accommodate future students.
- Candidate Key: StudentID
- Primary Key: Student ID
- Strong/Weak Entity: Strong
- Fields to be Indexed: StudentID, Name
- Attributes:

Name	**StudentID**	**Name**
Description	The student's ID number	The student's name
Domain/Type	Unsigned Integer	String
Value Range	0...2^32	Any
Default Value	None	None
Nullable	No	No
Unique	Yes	No
Single or Multiple value	Single	Single
Simple or Composite	Simple	Composite

Department

- This entity will be used to determine the student's disciplines. A student can theoretically have as many disciplines as he can.
- Candidate Keys: DepartmentID, DepartmentHead
- Primary Key: DepartmentID
- Strong/Weak Entity: Strong
- Fields to be Indexed: DepartmentID
- Attributes:

Name	**DepartmentID**	**DepartmentHead**	**Extension**
Description	The Department's ID	The name of the department head	The extension to the department's head office.
Domain/Type	String	String	Unsigned Integer
Value Range	Any	Any	1000-9999
Default Value	None	None	None
Nullable	No	No	No
Unique	Yes	Yes	Yes
Single or Multiple value	Single	Single	Single
Simple or Composite	Simple	Composite	Simple

Class

- The Class entity will comprise of the classes that are being offered at the school and the classes that the students can be tutored in.
- Candidate Key: ClassID
- Primary Key: ClassID
- Strong/Weak Entity: Strong
- Fields to be Indexed: ClassID
- Attributes:

Name	**ClassID**	**ClassDescription**
Description	The ID number of the class.	The description of the course.
Domain/Type	String	String
Value Range	Any	Any
Default Value	None	None
Nullable	No	No
Unique	Yes	Yes
Single or Multiple value	Single	Single
Simple or Composite	Simple	Simple

Instructor

- The Instructor entity contains information of the faculty of the university
- Candidate Key: InstructorID, InstructorName
- Primary Key: InstructorID
- Strong/Weak Entity: Strong
- Fields to be Indexed: ClassID
- Attributes:

Name	**InstructorID**	**InstructorName**	**Extension**
Description	The ID number of the class.	The description of the course.	The extension to the instructor's office.
Domain/Type	Unsigned Integer	String	Unsigned Integer
Value Range	0...2^32	Any	1000 - 9999
Default Value	None	None	None
Nullable	No	No	Yes
Unique	Yes	Yes	Yes
Single or Multiple value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Relationship Set Description

Tutors

Tutors is a binary relationship between the Tutor entity and the Subject entity. This relationship maps a tutor to a specific subject they are able to tutor.

- Mapping Cardinality: M..N
- Descriptive Field: None
- Participation Constraint: In order to be a tutor you must be able to tutor in one of the subject areas. Therefore there is total participation.

Works

Works is a relationship between the Tutor entity and the Timeblock entity. The relationships describes the amount of hours that a tutor works as well as the tutor's tutoring schedule.

- Mapping Cardinality: M..N
- Descriptive Field: None
- Participation Constraint: If you are not working any hours in the tutoring center then you are not a tutor, therefore the participation constraint is total.

Tutored

Tutored is a binary relationship between the Student entity, and the Class entity. This will keep track of which students a certain tutor has tutored, as well as the class tutored. This would provide clarity to which tutor is doing well. Also Tutored will keep track of the time a certain student was being tutored.

- Mapping Cardinality: M..N
- Descriptive Field: None
- Participation Constraint: This has partial participation due to the fact that student may forget to record their students id numbers and the time spent in the tutoring center.

Studying

Studying is a relationship between the student and the school's department. The relationship will inform us what the student's major is. This relationship will help the tutoring center have an idea on who they should focus on.

- Mapping Cardinality: M..N
- Descriptive Field: None
- Participation Constraint: This has total participation.

Taking

Taking is a ternary relationship between the Student entity, the Class entity, and the Instructor entity. This relationship will map a student to a class that he or she is taking. It will also give a class description and the instructor teaching the course, as well more information on the class itself.

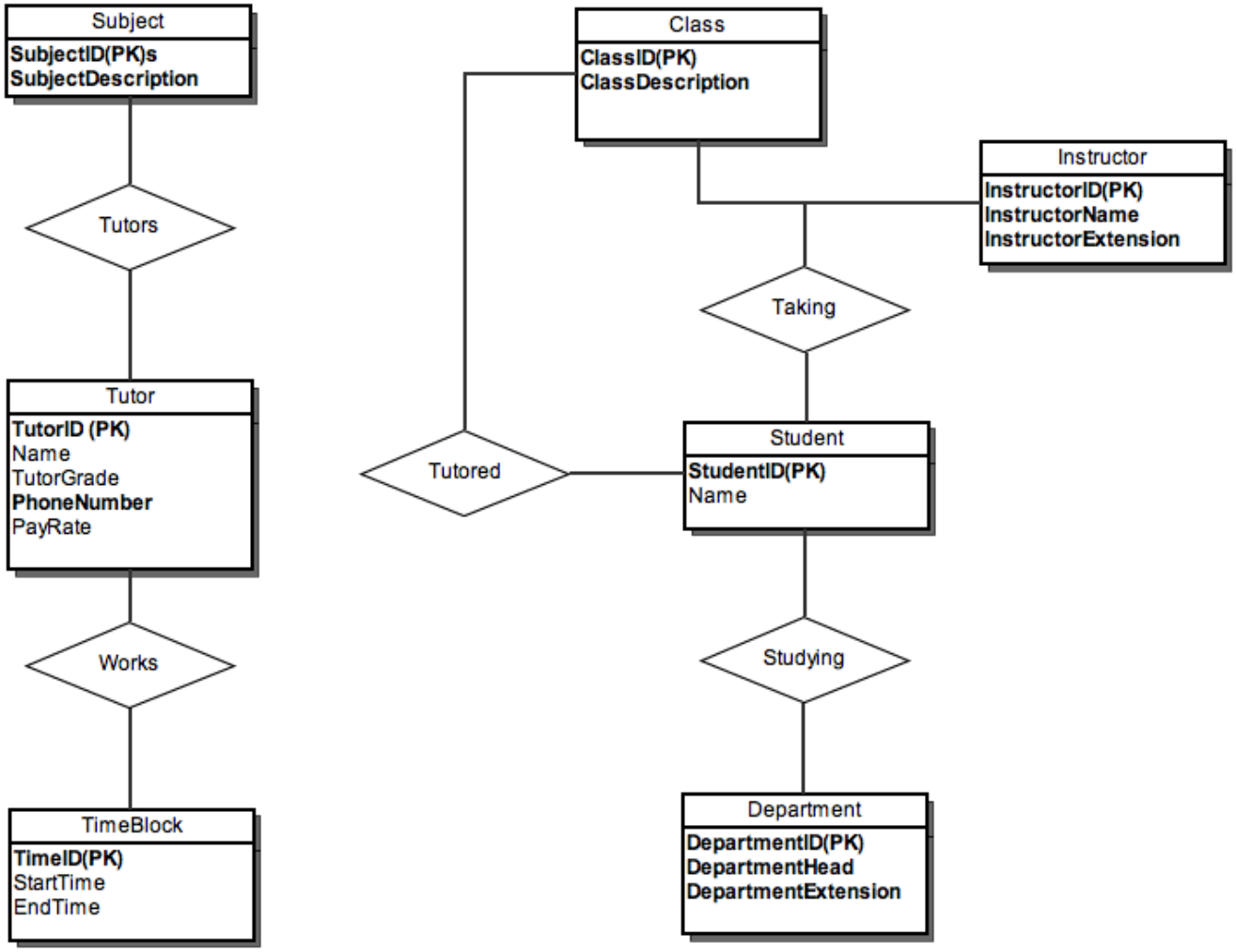
- Mapping Cardinality: M..N

- Descriptive Field: None
- Participation Constraint: This has total participation for the students.

Related Entity Set

Since this is a relatively small business there are no related entity sets. There is divide between the two major entities in this database and thus do not share any entities. If the database becomes more complex there then may be some related entity sets.

E-R Diagram


Subject
SubjectID(PK)s
SubjectDescription
Tutors
Tutor
TutorID (PK)
Name
TutorGrade
PhoneNumber
PayRate
Works
TimeBlock
TimeID(PK)
StartTime
EndTime
Class
ClassID(PK)
ClassDescription
Instructor
InstructorID(PK)
InstructorName
InstructorExtension
Taking
Tutored
Student
StudentID(PK)
Name
Studying
Department
DepartmentID(PK)
DepartmentHead
DepartmentExtension

E-R Model and the Relational Model

Description

The E-R Model just discussed recently helps in creating and planning out how a database should function. With that in mind, we can not simply use this model to implement the database, we must first convert the E-R Model to a Relational Model. The Relational Model provides a declarative method for specifying our data and queries. In other words this model allows us to state exactly what we want to gather from the database. For instance, if we had a database that kept track of baseball stats, we demand from the database a list of all pitchers who have an E.R.A (earned run average) less than 2.5, and the database will take that query and return us the results. As opposed to the E-R model, where if we gave that query it will not know how to find that information.

Comparison

The entity-relationship model is a conceptual model which allows us to design our database. Essentially it is a visual description of the database's format, attributes, relationships and its cardinality. But this model does not provide any details in actually implementing our database, and that is where the problem lies. The entity-relationship is a great first step in visualizing and designing our database, but the relational model is the next step of making the database an actuality.

The relational model on the other hand represents the entities and relationships in a table format with its attributes as the columns of the tables. Each row in the tables created by the relational model are tuples in the database, a tuple is a record of the database. The tables themselves represent a relationship in the model. All this information allows us to understand the size, constraints, and structure of the database more.

Conversion From E-R Model to Relational Model

To convert our E-R Model into a relational model we need follow an algorithm for each of our entities and our relationships. Our E-R model will provide the foundation of our relational model, meaning we will build off of the ideas and structure presented by the E-R Model to create our relational model.

The first step in converting our conceptual model to a relational model is to map the regular (strong) entities of the conceptual model. This is achieved by creating a relation using the same attributes from the entity from the E-R model. From here an attribute will be chosen to represent the primary key for the relation. If an attribute that is composite is chosen to be the primary key then a combination of those sections will be chosen as the primary key. If necessary a secondary key can be used as well.

The second step in the conversion will be to map all the weak entities in the E-R model. With the weak entities a relation will be created by using all the simple attributes of said entity. Then the

primary keys of the entity that owns the weak entity will be used as the foreign key for the relation. Finally the primary key for the weak entity will consist of the combination of the primary key(s) of the owner entity and the partial key of the weak entity, if any exist.

Next will be the mapping of all binary relationships will be handled. The relationships from the E-R model will be mapped to separate relations S and T. There are three methods to create this representation in the relational model:

- Include the primary key of one relation as a foreign key of another. This method is best used for total participation of one the entities.
- Merge both entity types into one relation. This method fits those relationships in which both entities exhibit total participation.
- Create a relation that holds all the primary keys of the entities participating in the relationship. This is necessary for relationships that have M:N cardinality. But this can still be used for relationships that do not have that.

The given steps are used to map all binary relationships that map from 1:1, 1:M, and M:N.

Next for each multivalued attribute you will need to create a relation for it. The relation will have an attribute for each section of the multivalued attribute. The relation will also contain, as the foreign key, the primary key of the entity. Then the primary key for the relation will be the combination of the foreign key and the attributes.

The final step is to convert any n-ary relationships you have in the E-R model, where $n > 2$, into a relation. So for any n-ary relationship type R, create a new relation S to represent the R. Include in S, as the foreign key, the primary keys of the entities involved in the relationship.

For conceptual models that involve specialization or generalization, more steps are needed in order to properly represent them in the relational model. There are several approaches for this step:

- Create one relation for the superclass with {k, a1, a2, …, an} attributes, and create m relations for each subclass with its own attributes unioned with the superclass' attributes, and specify k as the subclass' primary key. This works for any arrangement of specialization: total, partial, disjoint, or overlapping
- Create a relation for each subclass, and union the subclass' attributes with the superclass' attributes. This works for when every superclass object belongs to at least one subclass.
- Create a single relation with all subclass and superclass attributes unioned. This can create many NULL entries in the resulting tuples.

Constraints

A relation is a set of unique tuples where in each tuple contains the same attributes as one another, same meaning number and type not actual value. With that in mind there constraints to these entities. Entity constraints are used to ensure that there are unique elements of each tuple in the relation. Also constraints ensure that no primary key are left NULL. For foreign keys there are constraints that make sure that there is valid reference to any other tuples in the relation. There are also constraints that serve the purpose of the company or business, these are called business constraints and check constraints. These constraints are specific to each type of business.

E-R Database to Relational Database

Tutor Relation

Attributes:

- TutorID
  - Domain: Unsigned integer: 1 to 2^(32) – 1. Value must not be Null.
- Name
  - Domain: String. The string will hold the name of the tutor in the following format: “Firstname Lastname.” The string will be strictly for display purposes. Should not be Null.
- TutorGrade
  - Domain: Character. The character will hold the present grade of the tutor, in terms of how well the tutor tutors. Ranges from {A,B,C,D,F}. Value can not be Null.
- PhoneNumber
  - Domain: Integer. The phone number of a tutor will be stored as a 10 digit integer, where the first 3 digits represent the area code. Can be Null.
- PayRate
  - Domain: Double. This is the payrate for each tutor. Can not be Null.

Constraints:

- Primary Key: TutorID will be the Primary Key. This value can not be null and must be unique.
- Business Rule: Pay rate can not exceed 12.00 dollars per hour.

Candidate Keys:

- TutorID

TimeBlock Relation

Attributes

- TimeID
  - Domain: Unsigned Integer: 1 to 2^(32)-1. Value can not be Null.
- StartTime
  - Domain: String. Value can not be Null. The string will be written in the following format “DD HH:MM”, and follow a 24 hour time type.
- EndTime
  - Domain: String. Value can not be Null. The string will be written in the following format “DD HH:MM”, and follow a 24 hour time type.

Constraints:

- Primary Key: TimeID acts as the primary key. Can not be Null. Must be unique.
- Business Rule: StartTime has to be smaller than EndTime.

Candidate Keys:

- TimeID

Subject Relation

Attributes

- SubjectID
  - Domain: String. Value can be of any length. Can not be Null.
- Description
  - Domain: String. Value can range from any length. Can be Null. Will describe the subject.

Constraints:

- Primary Key: SubjectID is the primary key, thus it can not be null and must be unique.

Candidate Keys:

- SubjectID

Student Relation

Attributes

- StudentID
  - Domain: Unsigned Integer. Value can range from 0 to 2^(32)-1. Can not be null.
- Name
  - Domain: String. The string will hold the name of the student in the following format: “Firstname Lastname.” The string will be strictly for display purposes. Should not be Null.

Constraints:

- Primary Key: StudentID is the primary key, thus it can not be null and must be unique.

Candidate Keys:

- StudentID

Department Relation

Attributes

- DepartmentID
  - Domain: String. Can not be null.
- DepartmentHead
  - Domain: String. String can be of any size following the format “Firstname Lastname”. Can not be null.
- DepartmentExtension
  - Domain: Unsigned Integer. Ranges from 0 to 2^(32)-1. Can be null. Represents the extension to the department.

Constraints:

- Primary Key: Department is the primary key so it must be unique and can not be null.
- Business Rule: Department extension must be between 1000 and 9999.

Candidate Keys:

- DepartmentID

Class Relation

Attributes

- ClassID
  - Domain: String. Can not be NULL.
- ClassDescription
  - Domain: String. Value can range from any length. Can be Null. Will describe the class.

Constraints:

- Primary Key: ClassID is the primary key so it can not be null and must be unique.

Candidate Keys:

- ClassID

Instructor Relation

Attributes

- InstructorID
  - Domain: Unsigned Integer. 1 to 2^(32)-1. Can not be null.
- InstructorName
  - Domain: String. String can be of any size following the format “Firstname Lastname”. Can not be null.
- InstructorExtension
  - Domain: Unsigned Integer. Ranges from 0 to 2^(32)-1. Can be null. Represents the extension to the instructor's office.

Constraints:

- Primary Key: InstrutorID is the primary key so it can not be nullable and must be unique.
- Business Rule: InstructorExtension must be between 1000 and 9999.

Candidate Keys:

- InstructorID

Tutors Relation

Attributes

- TutorID
  - Domain: Unsigned integer: 1 to 2^(32) – 1. Value must not be Null.
- SubjectID

    - Domain: String. Value can be of any length. Can not be Null.

Constraints:

- Foreign Keys: The foreign keys TutorID and SubjectID must be a valid record from each relation.

Candidate Keys:

- None

Works Relation

Attributes:

- TutorID
    - Domain: Unsigned integer: 1 to 2^(32) – 1. Value must not be Null.
- TimeID
    - Domain: Unsigned Integer: 1 to 2^(32)-1. Value can not be Null.

Constraints:

- Foreign Keys: TutorID must be a valid record from the relation, as well as TimeBlockID.

Candidate Keys:

- None

Tutored Relation

Attributes:

- StudentID
    - Domain: Unsigned Integer. Value can range from 0 to 2^(32)-1. Can not be null.
- SignIn
    - Domain: String Value. Can be Null. The string will be written in the following format “MM/YY/DD HH:MM AM/PM”.
- SignOut
    - Domain: String Value. Can be Null. The string will be written in the following format “MM/YY/DD HH:MM AM/PM”.
- ClassID
    - Domain: String. Can not be null.

Constraints:

- Foreign Keys: TutorID, StudentID, and ClassID must be valid relations and exist.
- Business Rule: SignOut can not be smaller than SignIn.

Candidate Keys:

- None

Taking Relation

Attributes:

- StudentID
  - Domain: Unsigned Integer. Value can range from 0 to 2^(32)-1. Can not be null.
- ClassID
  - Domain: String. Can not be null.
- InstructorID
  - Domain: Unsigned Integer. 1 to 2^(32)-1. Can not be null.
- Grade
  - Domain: Character. The character will hold the present grade of the student, in terms of how well the student is doing in the class. Ranges from {A,B,C,D,F}. Value can not be Null.

Constraints:

- Foreign Keys: StudentID, ClassID, and InstructorID must exist in their respective relation.

Candidate Keys:

- None

Studying Relation

Attributes:

- StudentID
  - Domain: Unsigned Integer. Value can range from 0 to 2^(32)-1. Can not be null.
- DepartmentID
  - Domain: Unsigned Integer. Value can range from 0 to 2^(32)-1. Can not be null.

Constraints:

- Foreign Keys: StudentID and DepartmentID must be valid records from their respective relaitons.

Candidate Keys:

- None.

Relational Instances

Tutor(TutorID, Name, TutorGrade, PhoneNumber, PayRate)

TutorID	Name	TutorGrade	PhoneNumber	PayRate
1	Kevin Velado	A	(555)555-5555	12
2	Chuck Cupani	A	(555)555-5556	10.5
3	Vianey Leos	A	(555)555-5557	10.5
4	Daniel Betancourt	A	(555)555-5558	10.5
5	Derrick McKee	A	(555)555-5559	10.5
6	Walter Morales	A	(555)555-5560	10.5
6	Chris Gutierrez	A	(555)555-5561	10.5
7	John Doe	B	(555)555-5562	10.5
8	Susie Q	B	(555)555-5563	10.5
9	Doreteo Rivera	A	(555)555-5564	10.5
10	Jane Doe	C	(555)555-5565	10.5

TimeBlock(TimeID, StartTime, EndTime)

TimeID	StartTime	EndTime
1	Mon 08:00	Mon 9:30
2	Mon 09:30	Mon 10:45
3	Mon 10:45	Mon 12:15
4	Mon 12:15	Mon 13:45
5	Mon 13:45	Mon 15:00
6	Mon 15:00	Mon 17:00
7	Mon 17:00	Mon 18:30
8	Mon 18:30	Mon 20:00
9	Sun 13:00	Sun 17:00
10	Tues 08:00	Tues 11:00

Subject (SubjectID, Description)

SubjectID	Description
1	Computer Science
2	Mathematics
3	Biology
4	English
5	French
6	Physics
7	Nursing
8	Psychology
9	Spanish
10	Geology

Student (StudentID, Name)

StudentID	Name
1	Gina Gonzales
2	Nehemias Ulloa
3	Jason Cotton
4	John Doe
5	Don Passion
6	Bob Smith
7	Tom Wallace
8	Reggie Roberson
9	Vianey Leos
10	Chris G

Department (DepartmentID, DepartmentHead, DepartmentExtension)

DepartmentID	DepartmentHead	DepartmentExtension
1	Marc Thomas	1000
2	Javier Trigos	1001
3	Thomas Meyers	1002
4	Charles Lam	1003
5	Robert Smith	1004
6	Jane Doe	1005
7	Edgar Buenrostro	1006
8	Alven Diaz	1007
9	Dig Dug	1008
10	Pac Man	1337

Class (ClassID, ClassDescription)

ClassID	ClassDescription
CS 221	Introduction to C++
FR 101	Intro to French
PHY 221	Classical Physics
Math 101	Finite Mathematics
Math 191	Precalculus I
Math 192	Precalculus II
CS 295	Discrete Structures
Bio 100	Introduction to Biology
PSY 100	Introduction to Psychology
Math 300	Sets and Logic

Instructor (InstructorID, InstructorName, InstructorExtension)

InstructorID	InstructorName	InstructorExtension
1	Charles Lam	1003
2	David Gove	1123
3	Pac Man	1337
4	Marc Thomas	1000
5	Dig Dug	1008
6	Max Velado	1466
7	Aaron Rodgers	1212
8	Gregory Jennings	1385
9	Ryan Grant	3232
10	John Doe	4367

Tutors (TutorID, SubjectID)

TutorID	SubjectID
1	1
1	2
1	3
1	4
1	5
1	6
1	7
2	2
2	4
2	5
2	6
3	2
3	1
4	2
5	5
5	6

TutorID	SubjectID
6	1
7	2
7	3
8	4
8	5
8	6
9	7
9	2
10	4
12	5
12	6
13	2
13	1
14	2
15	5
16	6

TutorID	SubjectID
21	1
21	2
21	3
21	4
21	5
21	6
21	7
22	2
22	4
22	5
22	6
23	2
23	1
24	2
25	5
25	6

5	7
5	8
5	9
5	10

16	7
16	8
16	9
17	10

52	7
75	8
85	9
95	10

Works (TutorID, TimeID)

TutorID	TimeID
1	1
1	9
1	16
1	24
1	32
2	2
2	10
2	18
2	26
2	34
3	3
3	11
3	19
3	27
3	35
4	4
4	12
4	20
4	28
4	36

TutorID	TimeID
5	5
5	13
5	21
5	29
5	37
6	6
6	14
6	22
6	30
6	38
7	7
7	15
7	23
7	31
7	39
8	8
8	16
8	24
8	32
8	40

TutorID	TimeID
9	9
9	17
9	25
9	33
9	41
10	10
10	18
10	26
10	34
10	42
11	11
11	19
11	27
11	35
11	43
12	12
12	20
12	28
12	36
12	42

Tutored (StudentID, SignIn, SignOut, ClassID)

StudentID	SignIn	SignOut	ClassID
23	10/10/10 08:00 AM	10/10/10 08:45 AM	CS 221
42	10/10/10 08:05 AM	10/10/10 09:00 AM	CS 223
34	10/10/10 09:15 AM	10/10/10 10:00 AM	Math 140
23	10/10/10 09:15 AM	10/10/10 11:00 AM	Bio 100
11	10/10/10 09:15 AM	10/10/10 10:00 AM	Math 140
10	10/10/10 09:15 AM	10/10/10 10:00 AM	Math 140
23	10/10/10 10:30 AM	10/10/10 10:45 AM	English 101
11	10/10/10 10:30 AM	10/10/10 01:00 PM	FR 101
15	10/10/10 10:30 AM	10/10/10 12:00 PM	Math 191
16	10/10/10 10:30 AM	10/10/10 12:00 PM	Math 191
15	10/10/10 10:30 AM	10/10/10 12:00 PM	Math 191
34	10/10/10 03:00 PM	10/10/10 04:15 PM	Math 201
27	10/10/10 06:00 PM	10/10/10 08:00 PM	Math 202

Taking (StudentID, ClassID, InstructorID, Grade)

StudentID	ClassID	InstructorID	Grade
1	CS 221	21	A
2	CS 221	21	B
3	CS 221	21	A
4	CS 221	21	B
5	MATH 101	45	A
6	MATH 101	45	D
7	MATH 101	45	A
8	MATH 101	45	B
9	MATH 101	45	C
10	MATH 101	45	C
11	BIO 101	54	C
12	MATH 101	45	B
13	MATH 101	45	A

StudentID	ClassID	InstructorID	Grade
31	CS 221	21	A
32	MATH 101	45	A
33	CS 221	21	A
34	MATH 101	45	A
35	CS 221	21	A
36	BIO 101	54	A
37	CS 221	21	A
38	CS 221	21	A
39	BIO 101	54	A
40	CS 221	21	B
41	MATH 101	45	B
42	MATH 101	45	B
43	BIO 101		B

14	BIO 101	54	A
15	BIO 101	54	A
16	BIO 101	54	A
17	MATH 101	45	B
18	MATH 101	45	F
19	MATH 101	45	B
20	BIO 101	54	B
21	BIO 101	54	B
22	MATH 101	45	B
23	BIO 101	54	C
24	CS 221	21	B
25	CS 221	21	B
26	CS 221	21	B
27	CS 221	21	C
28	CS 221	21	A
29	CS 221	21	D
30	CS 221	21	A

44	CS 221	21	B
45	CS 221	21	B
46	CS 221	21	B
47	BIO 101	54	C
48	CS 221	21	B
49	CS 221	21	B
50	CS 221	21	B
51	CS 221	21	B
52	BIO 101	54	B
53	CS 221	21	B
54	CS 221	21	B
55	MATH 101	45	B
56	CS 221	21	B
57	MATH 101	45	B
58	CS 221	21	B
59	CS 221	21	B
60	MATH 101	45	B

Studying (StudentID, DepartmentID)

StudentID	DepartmentID
1	CS
2	BIO
3	MATH
4	PHY
5	CS
6	BIO
7	MATH
8	BIO
9	CS
10	CS
11	BIO

StudentID	DepartmentID
31	MATH
32	MATH
33	MATH
34	CS
35	CS
36	MATH
37	MATH
38	CJ
39	CJ
40	MATH
41	MATH

12	MATH
13	MATH
14	CS
15	BIO
16	CS
17	BIO
18	CS
19	BIO
20	MATH
21	BIO
22	CS
23	MATH
24	MATH
25	BIO
26	CS
27	MATH
28	CS
29	BIO
30	CS

42	PHY
43	MATH
44	MATH
45	CJ
46	BIO
47	BIO
48	MATH
48	GEO
49	BIO
50	MATH
51	GEO
52	BIO
53	BIO
54	MATH
55	CS
56	MATH
57	BIO
58	MATH
59	BIO

Queries

- List all students taking Math 140
- List students only taking Math 140
- List tutors that tutor at least 2 subjects
- List Instructors who are also Department Heads
- List all Physics majors who are being tutored

Query Representation

List All Students Taking Math 140

- Relational Algebra
  π (StudentID)(σ (Taking t1) * Student)
  t1.ClassID = "Math 140"
- Tuple Relational Calculus
  {s | Student(s) ^ (∃ t) (Taking(t) ^ s.StudentID = t.StudentID ^ t.ClassID = "Math 140")}
- Domain Relational Calculus
  {<s,n> | Student(s,n) ^ (∃ t)(Taking(t = s, "Math 140", –, –)) }

List Students Only Taking Math 140

- Relational Algebra
  π (StudentID)((π(StudentID)(σ(Taking t1)) -
  t1.ClassID = "Math 140"
  π (StudentID)(σ (Taking t2)))* Student)
  t2.ClassID != "Math 140"
- Tuple Relational Calculus
  {t | Taking(t) ^ t.ClassID = "Math 140" ^ ~(∃t2)(Taking(t2) ^ t.StudentID = t2.StudentID ^ t.ClassID != t2.ClassID)}
- Domain Relational Calculus
  { <s> | Taking(s,"Math 140", – , –) ^ ~(∃c)(Taking(s,c!= "Math140", – , –))}

List Tutors That Tutor at Least 2 Subjects

- Relational Algebra
  π(TutorID)(σ(Tutors t1 x Tutors t2) * Tutor)
  t1.TutorID = t2.TutorID
  ^ t1.SubjectID != t2.SubjectID
- Tuple Relational Calculus
  {t | Tutors(t) ^ (∃ t2)(Tutors(t2)^t.TutorID= t2.TutorID ^ t.SubjectID != t2.SubjectID)}
- Domain Relational Calculus
  {<t,s>| Tutors(t,s) ^ (∃t2)(∃s2)(Tutors(t2 = t, s2!=s))}

List All Instructors Who Are Also Department Heads

- Relational Algebra
  π(InstructorName)(σ(Instructor i x Department d) * Instructor)
  i.InstructorName = d.DepartmentHead

- Tuple Relational Calculus
  {i | Instructor(i) ^ ($\exists$ d)(Department(d) ^ d.DepartmentHead= i.InstructorName) }

- Domain Relational Calculus
  {<i> | Instructor(– , i , –) ^ ($\exists$ c) (Department(– , c = i , –))}

List All Physics Majors Who Are Being Tutored

- Relational Algebra
  π(Name)(σ(Tutored t x Studying s) * Student)
  t.StudentID = s.StudentID
  ^ s.Major = "PHYS"

- Tuple Relational Calculus
  { n | Student(n) ^ ($\exists$ t)($\exists$ s)(Tutored(t) ^ Studying(s) ^ t.StudentID = s.StudentID
  ^ s.StudentID = n.StudentID
  ^ s.Major = "Math 140") }

- Domain Relational Calculus
  {<s,n> | Student(s,n) ^ ($\exists$ x)($\exists$ y)(Tutored(x = s, – , – , –) ^ Studying(y = s,"Phys"))}

Phase III: Implementation of the Relational Database

SQLPLUS

Now that the relational model has been completed, the description for each relation can be used to actually create a database that meets its requirements regarding attributes, constraints, and relationships. To do this, I will use the implementation of SQL from the Oracle Relational Database Management System (hereafter referred to as Oracle). Structured Query Language, or SQL, was first developed at IBM in the 1970s. Since then, it has undergone rigorous optimization and standardization, and several popular implementations are used for most databases. These include Microsoft's Transact-SQL, or T-SQL, MySQL, and Oracle. Oracle provides a tool called SQL*PLUS that allows users to interactively run any SQL commands. It's a command-line tool that supports both user interaction and automated scripts. By using a hierarchy of scripts to call appropriate commands, a database can be destroyed and re-created very quickly using SQL*PLUS.

Schema Objects in Oracle

Oracle has a collection of schema objects objects that create schema. The tablespace organizes the structutre of the database taking into consideration the different kinds of schemas. As well as containing the loactions used to store data on the database. Schema objects are data structures that are stored in a given database, logically. Oracle has the following schema objects:

Tables

Tables represent each relation from the relational model. Tables are the basic storage system used in an Oracle database. The columns on a table represent the attributes of that relation. Each column has a unique name and data type for that relation. The rows represent a valid record in that relation with certain unique attributes. Tables also hold information about the relation's primary keys, foreign keys, and constraints.

Views

Views act like read-only commands that return tuples from tables that meet some requirements. Views allow for better organization and optimization of the database. The output of a view can be seen as a table in a sense, thus can be accessed and modified like one. Unlike tables, views will not keep track of referential or integrity constraints. The results of a view are not stored anywhere, they are only simply displayed. Therefore views do not use up any storage space. Views can be used to obfuscate data, prevent direct access for certain users, and simplify representation for users. Materialized views are special views that perform a specific function on the data it retrieves, including aggregate functions, sorting, summations, data transfer, and reorganization.

Sequences

Sequence generators create a sequential set of numbers for use in a multi-user environment. These sequence numbers can then be used to determine order for queued operations or requests. They are not dependent on any table, but they can be used to generate primary keys for a specific table. Sequence numbers can also be used to keep track of roll-backs in transactions, ensuring that the right commands are reversed without confliction between separate users.

Synonyms

Synonyms are alternate aliases for certain types of schema objects, such as tables, procedures, functions, or views. They do not require any additional storage space other than their entries in the database's data dictionary. Synonyms can be used to directly hide internal data for outside users or to simplify complex SQL commands.

Indexes

Databases attempt to optimize traversal of each table by caching the values of unique attributes, such as primary keys. Additional attributes can be specified such that the database more quickly accesses their values during comparisons for overall faster results. Indexes can also be created for combinations of certain attributes. Furthermore, an existing index can be used to create another dependent index. An Oracle system will automatically maintain indexes once specified by a user.

Database Links

Put simply, database links are hard-coded, read-only links to another database. This allows one database to perform queries and retrieve results using another database, while simultaneously preventing both databases from risking the integrity of one another.

Stored procedures and functions

These can be seen as scripts that are stored on the database. When executed, a stored procedure or function always performs the same task as instructed upon its creation. Functions in Oracle always return a single value to the user, while stored procedures do not.

Packages

Packages are a specific collection of stored procedures, functions, and cursors. Combined, they act as a single unit of instructions. This is critical for large-scale operations performed by stored procedures. Packages organize and simplify design requirements for databases that require persistent, complex tasks.

Schema objects in this project

In this project, the two most frequently used schema objects are the table and the view.
Most of the tables are created using syntax similar to this:

```
CREATE Table [TableName]
(
        attributes      attribute types         null? ,
        …               …                       … ,

        CONSTRAINT pk_tablename PRIMARY KEY (AttributeName),
        CONSTRAINT k_ParentName_ChildName FOREIGN KEY (AttributeName)
                REFERENCESParentName (ParentAttributeName)

);
```

The scheme objects created using this syntax are as follows:

- kv_tutor Tutor relation
- kv_timeblock Timeblock relation
- kv_subject Subject relation
- kv_student Student relation
- kv_department Department relation
- kv_class Class relation
- kv_instructor Instructor relation
- kv_tutors Tutors relation
- kv_works Works relation
- kv_tutored Tutored relation
- kv_taking Taking relation
- kv_studying Studying relation

Following are the schemas and instances for each relation:

kv_tutor

```
CS342 SQL> desc kv_tutor;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 TUTORID                                   NOT NULL NUMBER(5)
 TUTORNAME                                 NOT NULL VARCHAR2(20)
 TUTORGRADE                                NOT NULL VARCHAR2(1)
 PHONENUMBER                               NOT NULL VARCHAR2(13)
 PAYRATE                                   NOT NULL NUMBER(4,2)

CS342 SQL> select  * from kv_tutor;

  TUTORID TUTORNAME           T PHONENUMBER      PAYRATE
---------- -------------------- - ------------ ----------
         1 Kevin Velado         A 555-555-5555         12
         2 Chuck Cupani         A 555-555-5556          2
         3 Vianey Leos          A 555-555-5557       10.5
         4 Daniel Betancourt    A 555-558-5555       10.5
```

```
         5 Derrick McKee        A 555-555-5559        10.5
         6 Walter Morales       A 555-555-5550        10.5
         7 Chris Gutierrez      A 555-555-5551       10.75
         8 Susie Q             A 555-555-5552         9.5
         9 Doreteo Rivera       A 555-555-5553        11.5
        10 Jane Doe             C 555-555-5554       10.25

10 rows selected.
```

kv_timeblock

```
CS342 SQL> desc kv_timeblock;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 TIMEID                                                    NOT NULL NUMBER(5)
 STARTTIME                                                 NOT NULL VARCHAR2(20)
 ENDTIME                                                   NOT NULL VARCHAR2(20)

CS342 SQL> select * from kv_timeblock;

    TIMEID STARTTIME           ENDTIME
---------- -------------------- --------------------
         1 Sun 13:00            Sun 18:00
         2 Mon 08:00            Mon 09:30
         3 Mon 09:30            Mon 10:45
         4 Mon 10:45            Mon 12:15
         5 Mon 12:15            Mon 13:45
         6 Mon 13:45            Mon 15:00
         7 Mon 15:00            Mon 17:00
         8 Mon 17:00            Mon 18:30
         9 Mon 18:00            Mon 20:00
        10 Tue 08:00            Tue 09:30

10 rows selected.
```

kv_subject

```
CS342 SQL> desc kv_subject;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 SUBJECTID                                                 NOT NULL NUMBER(5)
 DESCRIP                                                   NOT NULL VARCHAR2(20)

CS342 SQL> select * from kv_subject;

 SUBJECTID DESCRIP
---------- --------------------
         1 Computer Science
         2 Math
         3 Biology
         4 English
         5 French
         6 Physics
         7 Nursing
         8 Psychology
         9 Spanish
        10 Geology

10 rows selected.
```

kv_student

```
CS342 SQL> desc kv_student;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 STUDENTID                                                 NOT NULL NUMBER(9)
 STUDENTNAME                                               NOT NULL VARCHAR2(20)

CS342 SQL> select * from kv_student;

 STUDENTID STUDENTNAME
---------- --------------------
         1 Gina Gonzales
         2 Nehemias Ulloa
         3 Jason Cotton
         4 John Doe
         5 Don Pasion
         6 Bob Smith
         7 Tom Wallace
         8 Reggie Roberson
         9 Vianey Leos
        10 Chris G
```

10 rows selected.

kv_department

```
CS342 SQL> desc kv_department
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 DEPARTMENTID                                              NOT NULL VARCHAR2(10)
 DEPARTMENTHEAD                                            NOT NULL VARCHAR2(20)
 DEPARTMENTEXT                                             NOT NULL NUMBER(4)

CS342 SQL> select * from kv_department;

DEPARTMENT DEPARTMENTHEAD      DEPARTMENTEXT
---------- ------------------- -------------
CS         Marc Thomas                 1000
MATH       Javier Trigos               1001
BIO        Todd McBride                1002
PHYS       Thomas Meyer                1003
NUR        Norman Keltner              1004
MDRNLAN    Teresa Fernandez            1005
ENG        Sophia Adjaye               1006
BUS        Mary Doucet                 1007
CJ         Robert Fong                 1008
PSY        Steve Bacon                 1009

10 rows selected.
```

kv_class

```
CS342 SQL> desc kv_class;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 CLASSID                                                   NOT NULL VARCHAR2(15)
 CLASSDESCRIP                                              NOT NULL VARCHAR2(30)

CS342 SQL> select * from kv_class;

CLASSID         CLASSDESCRIP
--------------- ----------------------------
CS 221          Introduction to C++
FR 101          Introduction to French
PHY 221           Classical Physics
MATH 101          Finite Mathematics
MATH 191          Precalculus I
MATH 192          Precalculus II
CS 295          Discrete Structures
```

```
BIO 101        Introduction to Biology
PSY 100        Introduction to Psychology
MATH 300       Sets and Logics
MATH 140       Intro to Statistics

11 rows selected.
```

kv_instructor

```
CS342 SQL> desc kv_instructor
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 INSTRUCTORID                              NOT NULL NUMBER(5)
 INSTRUCTORNAME                            NOT NULL VARCHAR2(20)
 INSTRUCTOREXT                             NOT NULL NUMBER(4)

CS342 SQL> select * from kv_instructor;

INSTRUCTORID INSTRUCTORNAME       INSTRUCTOREXT
------------ -------------------- -------------
           1 Marc Thomas                  1000
           2 Javier Trigos                1001
           3 Charles Lam                  1122
           4 David Gove                   1123
           5 Pac Man                      1337
           6 Dig Dug                      1238
           7 Donna Meyers                 2022
           8 Thomas Meyer                 1003
           9 Ryan Grant                   3225
          10 Aaron Rodgers                1212

10 rows selected.
```

kv_tutors

```
CS342 SQL> desc kv_tutors;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 TUTORID                                   NOT NULL NUMBER(5)
 SUBJECTID                                 NOT NULL NUMBER(5)

CS342 SQL> select * from kv_tutors;

   TUTORID  SUBJECTID
---------- ----------
         1          1
         1          2
         2          2
         3          2
```

```
         4          4
         4          5
         4          9
         5          6
         6          1
         6          2
         6          3
         6          4
         6          5
         6          6
         6          7
         6          8
         6          9
         6         10
         7         10
         8          3
         9          3
        10          2

22 rows selected.
```

kv_works

```
CS342 SQL> desc kv_works;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------------------------------------------
 TUTORID                                                                 NOT NULL NUMBER(5)
 TIMEID                                                                  NOT NULL NUMBER(5)

CS342 SQL> select * from kv_works;

   TUTORID     TIMEID
---------- ----------
         1          2
         1          7
         1         10
         2          3
         2          4
         3          1
         3          5
         4          1
         5          6
         6          7
         7          8
         8          9
         9         10
        10          5
```

14 rows selected.

kv_tutored

```
CS342 SQL> desc kv_tutored;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 STUDENTID                                                 NOT NULL NUMBER(9)
 CLASSID                                                   NOT NULL VARCHAR2(15)
 SIGNIN                                                    NOT NULL DATE
 SIGNOUT                                                   NOT NULL DATE

CS342 SQL> select * from kv_tutored;

 STUDENTID CLASSID        SIGNIN    SIGNOUT
---------- --------------- --------- ---------
         1 CS 221          10-OCT-10 10-OCT-10
         2 BIO 101         13-OCT-10 13-OCT-10
         2 CS 221          10-OCT-10 10-OCT-10
         3 CS 221          10-OCT-10 10-OCT-10
         4 CS 221          11-OCT-10 11-OCT-10
         4 FR 101          11-OCT-10 11-OCT-10
         5 MATH 140        12-OCT-10 12-OCT-10
         5 MATH 140        13-OCT-10 13-OCT-10
         5 MATH 140        13-OCT-10 13-OCT-10
         6 MATH 140        12-OCT-10 12-OCT-10
         7 MATH 140        12-OCT-10 12-OCT-10
         8 BIO 101         13-OCT-10 13-OCT-10
         8 MATH 140        12-OCT-10 12-OCT-10
         9 CS 295          13-OCT-10 13-OCT-10
        10 MATH 140        13-OCT-10 13-OCT-10
        10 PSY 100         13-OCT-10 13-OCT-10

16 rows selected.
```

kv_taking

```
CS342 SQL> desc kv_taking;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 STUDENTID                                                 NOT NULL NUMBER(9)
 CLASSID                                                   NOT NULL VARCHAR2(15)
 INSTRUCTORID                                              NOT NULL NUMBER(5)
 CLASSGRADE                                                NOT NULL VARCHAR2(1)

CS342 SQL> select * from kv_taking;

 STUDENTID CLASSID        INSTRUCTORID C
```

```
---------- -------------- ------------ -
         1 CS 221                  1 A
         2 BIO 101                  5 B
         2 CS 221                  1 C
         3 CS 221                  7 A
         4 CS 221                  1 B
         4 FR 101                  6 C
         5 MATH 140                   4 C
         6 MATH 140                   3 A
         7 MATH 140                   4 D
         8 BIO 101                  5 C
         8 MATH 140                   3 B
         9 CS 295                  7 B
        10 MATH 140                    3 A
        10 PSY 100                  10 A

14 rows selected.
```

kv_studying

```
CS342 SQL> desc kv_studying
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------------------------------------------
 STUDENTID                                                               NOT NULL NUMBER(9)
 DEPARTMENTID                                                            NOT NULL VARCHAR2(10)

CS342 SQL> select * from kv_studying;

 STUDENTID DEPARTMENT
---------- ----------
         1 CS
         1 MATH
         2 CJ
         3 BIO
         4 BUS
         5 PHYS
         6 NUR
         7 MDRNLAN
         8 ENG
         9 PSY
        10 CS

11 rows selected.
```

SQL Queries

List all students taking math 140:

```
select unique s.studentID, s.studentName from kv_taking t, kv_student s
        where (s.studentID = t.studentID and t.classID = 'MATH 140')
        order by s.studentID
        /
```

```
CS342 SQL> @tutor_query01.sql;

 STUDENTID STUDENTNAME
---------- --------------------
         5 Don Pasion
         6 Bob Smith
         7 Tom Wallace
         8 Reggie Roberson
        10 Chris G
```

List all students that are only taking math 140:

```
select unique s.studentID, s.studentName from kv_taking t, kv_student s
        where (s.studentID = t.studentID and t.classID = 'MATH 140'
        and not exists ( select s.studentID, s.studentname from kv_taking t2
        where(t2.classID <> 'MATH 140' and t2.studentID = s.studentID ) ) )
        order by s.studentID
        /
```

```
CS342 SQL> @tutor_query02.sql;

 STUDENTID STUDENTNAME
---------- --------------------
         5 Don Pasion
         6 Bob Smith
         7 Tom Wallace
```

List tutors that tutor at least 2 subjects:

```
select unique t.tutorid, t.tutorname from kv_tutor t, kv_tutors s, kv_tutors s2
        where (t.tutorid = s.tutorid and s2.tutorid = t.tutorid
        and s2.subjectid <> s.subjectid) order by t.tutorid
        /
```

```
CS342 SQL> @tutor_query03.sql;
```

TUTORID TUTORNAME
--------- --------------------
1 Kevin Velado
4 Daniel Betancourt
6 Walter Morales

List all instructors who are also department heads:

```
select unique i.instructorid, i.instructorname from kv_instructor i, kv_department d
        where (d.departmenthead = i.instructorname) order by i.instructorid
        /
```

CS342 SQL> @tutor_query04.sql;

INSTRUCTORID INSTRUCTORNAME
----------- --------------------
1 Marc Thomas
2 Javier Trigos
8 Thomas Meyer

List all physics majors who are being tutored:

```
select unique s.studentid, s.studentname from kv_student s, kv_tutored t, kv_studying st
        where(t.studentid = st.studentid and st.studentid = s.studentid
        and st.departmentid = 'PHYS')
        /
```

CS342 SQL> @tutor_query05.sql;

STUDENTID STUDENTNAME
---------- --------------------
5 Don Pasion

List the number of subjects each tutor tutors:

```
select tutorid, count(subjectid) from kv_tutors group by tutorid
        /
```

CS342 SQL> @tutor_query06.sql;

TUTORID COUNT(SUBJECTID)
--------- ----------------
1 2
2 1
3 1
4 3
5 1
6 10
7 1
8 1

9 1
10 1

10 rows selected.

List the number of subjects for tutors who tutor at least 2 subjects:

```
select tutorid, count(subjectid) from kv_tutors
        group by tutorid having count(subjectid) > 1
        /
```

```
CS342 SQL> @tutor_query07.sql;

  TUTORID COUNT(SUBJECTID)
--------- ---------------
        1               2
        4               3
        6              10
```

List all students who have an A in math 140:

```
select unique s.studentid, t.classgrade from kv_student s, kv_taking t
        where (t.classid = 'MATH 140' and t.studentid = s.studentid
        and t.classgrade = 'A')
        /
```

```
CS342 SQL> @tutor_query08.sql;

 STUDENTID C
---------- -
         6 A
        10 A
```

List all departments who do not have a department chair who is also currently teaching:

```
select unique departmentid from kv_instructor outer join kv_department
        on (departmenthead != instructorname)
        /
CS342 SQL> @tutor_query09.sql;

DEPARTMENT
----------
PHYS
NUR
MDRNLAN
BIO
BUS
ENG
```

CS
MATH
CJ
PSY

10 rows selected.

Phase IV: Stored Subprograms, Packages and Triggers

Common Features in Oracle PL/SQL and MS Trans-SQL

MS trans-sql and oracle pl/sql have similar implementations of SQL. Both languages have the following a method to create tables, constraints, functions, stored procedures, triggers and packages. The most obvious difference between the two languages is of course the language structure. By this I mean the language used to implement the commands listed have different syntax.

The differences between MS and Oracle SQL are usually due to the fact that there are many different versions of Oracle SQL. In Trans-SQL it is not possible to update a huge amount of records, while Oracle does support this. Stored procedures are supported in both PL/SQL and Trans-SQL, so depending on the database it would be wise to implement procedures. Stored procedures are useful to perform basic SQL commands, and make it less tedious to the programmer to insert, delete, or update any records in the database. Some more positives from using stored procedures is that it will allow the programmer to hide any important tasks from the user, thus making the database more protected from their users.

Oracle PL/SQL

PL/SQL have three main sections in their programs Declaration, Execution, and Exception. Declarations is the section where variables, cursors, and user defined exceptions are declared or created. Executions is where the SQL commands are that perform the program's intent. Exception is where all the exception handling is performed, both user defined and system defined exceptions.

Most PL/SQL procedures have the following general structure:

```
DECLARE
        variableName        variableType        := value;
                                    .
                                    .
                                    .
BEGIN
        *SQL Commands go here (SELECT,INSERT,UPDATE,DELETE etc.)*
                                    .
                                    .
                                    .
END;
```

You can use all the variable types that are supported by Oracle are supported in PL/SQL.

Cursors are used to traverse a table, essentially a cursor is able to store tuples. Cursors are defined by

doing the following:

```
DECLARE
        CURSOR name (parameter_list)
        IS select_statement;
```

Control statements are supported in PL/SQL procedures. IF ELSE IF, FOR loops, and WHILE loops are supported in PL/SQL procedures. They follow the following format:

```
IF condition THEN statement;
ELSEIF condition THEN statement;
END IF;
LOOP
        EXIT WHEN can be used to quit this loop
END LOOP;
FOR I IN lower .. upper LOOP
        statement
END LOOP;
```

Exception handling follow the following format:

```
DECLARE
        * the procedures declarations *
BEGIN
IF condition THEN RAISE exception;
        END IF;
EXCEPTION
        WHEN exception_name THEN statement;
END;
```

Stored procedures can greatly facilitate performing complex jobs on the SQL server's data while maintaining abstraction for non-technical users. The structure of a stored procedure depends on the type what you want to perform, yet the syntax is the similar:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ (variablename                IN|OUT                variabletype)]
AS
(DECLARE variables go here)
BEGIN
        SQL statements
END;
```

Execution of a stored procedure from SQL*PLUS can be accomplished as follows:

```
SQL> exec procedure_name(arguments);
```

Stored functions are created and run in a method similar to stored procedures. But, stored functions declare a variable type that they will return. These can be used to guarantee that a variable will be returned. The syntax is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name
[ (variablename          IN|OUT          variabletype)]
RETURN datatype;
AS
(DECLARE variables go here)
BEGIN
        SQL statements;
        RETURN variable;
END;
```

Packages are a distinct collection of procedures and functions very similar to classes in C/C++. Creating a package requires a prototype for each included procedure and function:

```
CREATE PACKAGE package_name AS
        PROCEDURE names..;
        FUNCTION names…;
END package_name;
CREATE PACKAGE BODY package_name AS
        PROCEDURE name IS…
        BEGIN
                Statements
        END;
        FUNCTION name RETURN DATATYPE IS…
        BEGIN
                Statements
                RETURN variable
        END;
END package_name;
```

Triggers make collecting records and logs easy to manage. The alternative would be to make the database manager to manually run commands when a certain action was performed on the database. A trigger would run the commands when the condition was met. Since these tasks are automated after the trigger's creation, the user does not have to worry about maintaining or checking data before or after the operations.

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE|AFTER          INSERT|DELETE|UPDATE OF COL [column_name] [OR DELETE|
UPDATE|INSERT]
ON table_name
DECLARE
          variables
BEGIN
        FOR EACH ROW
        [WHEN CONDITION]
        Statements;
```

```
END;
```

Oracle PL/SQL Subprograms

I created 2 procedures, 1 function, and 1 trigger for this database. They are as follow:

Procedure 1: kv_deleteTutor.sql
This stored procedure simply takes in the number passed into the argument and deletes all records associated with the tutor id.

```
CREATE OR REPLACE PROCEDURE kv_deleteTutor(Tutor_ID IN number)
	AS
	BEGIN
		delete from kv_works
		where tutorID = Tutor_ID;

		delete from kv_tutors
		where tutorID = Tutor_ID;

		delete from kv_tutor
		where tutorID = Tutor_ID;

EXCEPTION
	when others then
	  raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
			'-ERROR-' || SQLERRM );

	END kv_deleteTutor;
/
```

Procedure 2: kv_insertTutored.sql
This procedure takes in the arguments needed to add a valid record into Tutored table.

```
CREATE OR REPLACE PROCEDURE kv_insertTutored(
	studentID IN number,
	classID IN varchar,
	signIn IN varchar,
	signOut IN varchar)
	AS
	BEGIN
		insert into kv_tutored values(
			studentID,
			classID,
			to_timestamp(signIn,'MM/DD/YY HH24:MI'),
			to_timestamp(signOut,'MM/DD/YY HH24:MI'));
	EXCEPTION
```

```
        when others then
          raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
                        '-ERROR-' || SQLERRM );
        END kv_insertTutored;
/
```

Function 1: topNAvgPayrate.sql
This function, when called, will calculate the average of the top 'n' pay rates of the tutors.

```
CREATE OR REPLACE FUNCTION kv_topNAvgPayrate( n IN NUMBER) RETURN NUMBER IS
        s NUMBER(9,2) := 0.0;
        p NUMBER(4,2) ;
        CURSOR c1 IS SELECT payRate FROM kv_tutor
                ORDER BY payRate DESC;
BEGIN
        open c1;
        FOR i IN 1..n LOOP
                fetch c1 into p;
                s := s+p;
        END LOOP;
        CLOSE c1;
        RETURN s/n;
EXCEPTION
        when others then
       raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
                        '-ERROR-' || SQLERRM );
END kv_topNAvgPayrate;
/
```

Trigger 1: kv_PhoneTrigger.sql
This trigger will fire off when the Phone number of a tutor is updated in any fashion. When done so the trigger will write the information into a log file.

```
CREATE OR REPLACE TRIGGER kv_phone_afterupdate
after update of phoneNumber on kv_tutor
for each row
BEGIN
        insert into kv_tutorLog
        values(tutor_log_sequence.nextval, sysdate, :old.tutorID, :old.tutorName, :old.phoneNumber,
:new.phoneNumber, :old.payRate);
END;
/
```

Phase V: GUI Design and Implementation

Description of Program and Its Daily Use

There are two applications essential for the organization of the Tutoring Center. The first is the management of the tutor's personal information and the statistics of the tutoring center (meaning how many students came in for tutoring, what class was most popular, etc.). The use of this application is limited due to the fact that updates to tutor personal information would be mainly done during the beginning of the year and sparsely throughout the year. Also printing out statements that show the statistics of the tutoring center are not needed until after current school period.

The second application for the Tutoring Center is the signing in system. The way the sign in system works is that the student will walk into the tutoring center and then input his information before he or she takes a seat. At the log in screen there are three options. They are as follows: New Student, Log In, and Log Out. These options are essential for the tutoring center. New Student will add a new record to the student database and keep track of that student's statistics. Log In and Log Out will verify the fact that tutoring center is in fact receiving traffic flow, and statistics from the log in system will be used to show how well the tutoring center is working out. This application is the most important of the two and is also used on a daily basis. Since this application was the most important this is the application I chose to create.

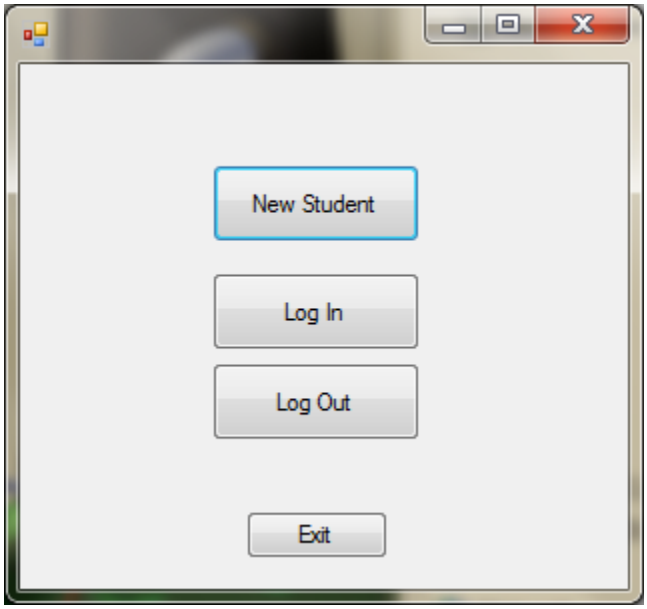
Relations, Views, and Subprograms Related

The first application is heavy with the amount of views, and subprograms used for it. But seeing that I did not code that program I am not sure which specific views and subprograms would be used. I could tell you that in general there would be a need to have a view that lists out the tutor's personal information, specialization, and work schedule. As well as a subprogram that will printout the tutoring center's statistics and other specific needs (e.g. which major comes in the most, what subject is being tutored the most, etc,).

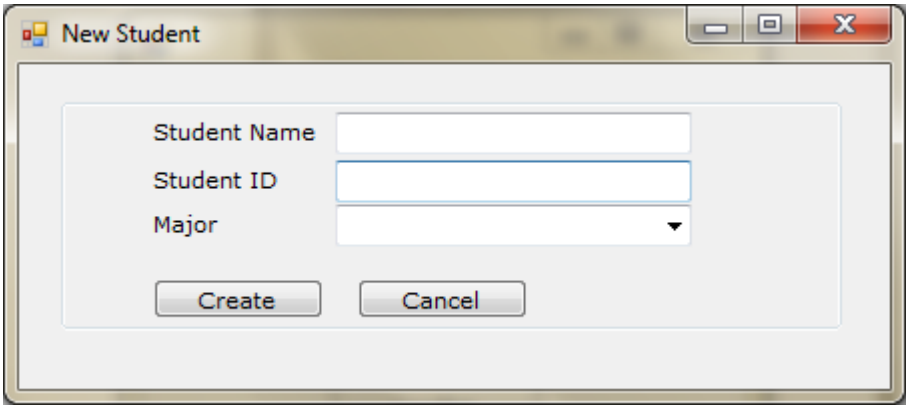
The second application can use a few views. The views necessary would be to show which majors, classes, and instructors the students can choose from. Besides that there are no other needs. Same goes with subprograms. The idea behind the second application is to be simple and hidden. Meaning the students who sign in and sign out should not see more information than they need and they should not be burdened with options.

Screen Shots and Description

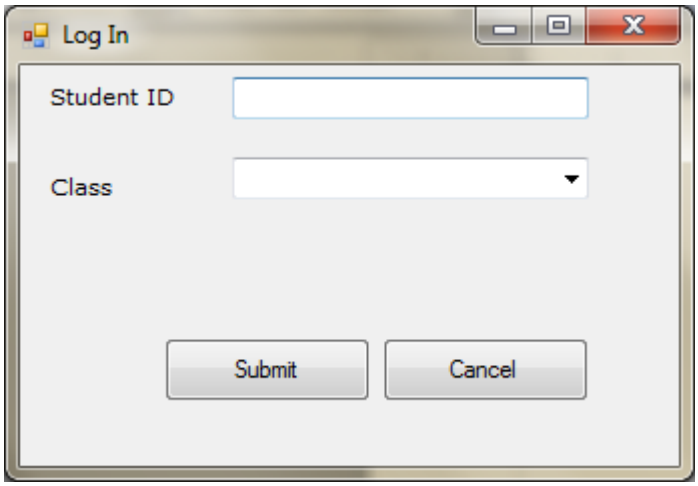
Since I only coded the second application I can only show screen shots on that application.

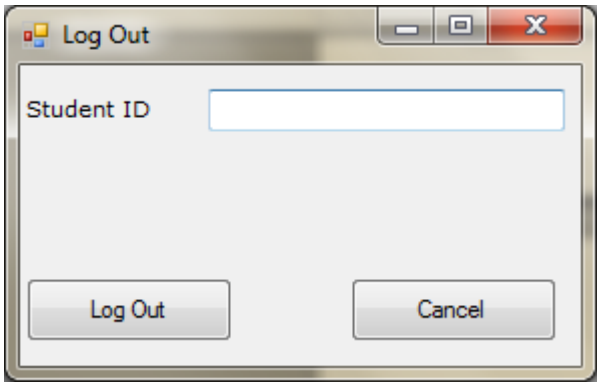


This first image shows the main menu for the second application of the Tutoring Center. As you can see simplicity is the key for this application. If the student is entering the tutoring center for the first time they will choose the option of New Student and create a record and unique ID for themselves. If they are returning to the tutoring center they will simply click on log in and enter their information. And regardless if it is the student's first time or not they will click on log out when they leave the tutoring center. Exit is simply there for debugging and should not be included in the final product as a visible obvious option.



This image shows what the student needs to input in order to create an valid id for the application. For now the information needed is bare, but in the final product it should ask for which classes they are currently taking and their grade and other information. Also to keep things simple all these decisions will be options shown to them in drop down menus. After they enter their information and click submit the application will return to the main menu and create an ID for that student to sign in with. Cancel will simply return to the main menu and not create a record for that student.





These two images show the Log In and Log Out systems. To Log In all you have to do is simply enter your ID number and choose which class you are taking from the drop down menu. In this iteration the classes are added to the drop down menu manually, hopefully in the future the menu will be created by reading it from the Class table from the database. Also to simplify searching adding a subject box will narrow down the results. The log out system is fairly simple, all you need to do is to enter you ID number and choose the option to log out.

What I Learned

I learned that creating a GUI is a very difficult task. There are two major and equal aspects to creating the GUI. The first is creating the lay out of your application. This step takes time and hard work to create. The second aspect to creating a GUI is the code. This aspect of the GUI is interesting due to the fact that coding for a GUI is very different than creating a console based program. This amplifies when you are to use a Database as a main feature in the application. The largest hurdle in this step was properly connecting to the database in order to use its features. All in all I'm glad I had this experience and hopefully this will prepare me in the future if I was ever tasked to create a GUI.