California Aid

FINANCIAL AID DATABASE

Omar Oseguera and Erick Ortiz | CMPS 3420 | Fall 2016

TABLE OF CONTENTS

[This page intentionally left blank]

1.1 Fact-Finding Techniques and Information Gathering

1.1.1 Introduction to Enterprise/Organization:

California Aid (CA) is a fictional Financial Aid service designed from financial aid organizations *California Student Opportunity and Access Program (Cal SOAP)* and *Free Application for Federal Student Aid (FAFSA)*. *CA* financial aid services are responsible for managing student financial assistance programs. These programs provide grants, loans, and work-study funds to students attending higher education. The goal of financial aid is to inform students and families about the availability of financial aid programs through outreach. This *Outreach* department will visit schools to teach students and families about the process of applying for and receiving aid. *Students* are responsible for submitting applications, which are updated by the *Logistics* department. All data is reviewed by a *Review Board* department which determines who gets Aid based on a *Budget*. *Financial Aid Packages* are then disbursed to students based on the *Budget*.

1.1.2 DESCRIPTION FACT-FINDING TECHNIQUES

In order to have a clear conceptual database, it is important to have a clear understanding of the data that will be stored in the database and how members of the organization will interact with it. The best method to acquire this understanding is to interview potential users of the database, that is, people working for the organization. To gather information for our database, we visited financial aid websites such as the Free Application for Federal Student Aid (FAFSA) and the California Student Opportunity and Access Program (Cal SOAP). Additionally, group member Omar Oseguera worked in Data Entry with the CSUB Cal SOAP offices. We also interviewed some of Omar's former coworkers on their daily tasks. The driving force behind such an organization is the combination of Outreach employees visiting High Schools to get Forms filled out by students, and Logistics employees entering data from new applicants.

1.1.3 SCOPE OF ENTERPRISE FOR CONCEPTUAL DATABASE:

In order to create our database, we need a clear understanding of the part of Financial Aid our database will represent. We will be designing our Financial Aid Database with a focus on the methods of a Cal SOAP branch. Cal SOAP's operations consists of an *Outreach* team visiting High Schools, a *Data Entry* team inputting information. The focus of our database will be to create a system keeping track of the interactions between *Students*, *Outreach*, and *Data Entry* in order to have a well-informed *Review Board* determine who gets financial Aid. The scope of our Database will be on the service for Students over internal tasks. This means we will not include information about internal meetings or any other type of decision making that does not affect the student.

1.1.4 ITEMIZED DESCRIPTIONS OF ENTITY SETS AND RELATIONSHIP SETS:

After establishing our fact-finding techniques, and determining the scope of our conceptual database, we are able to present our data as Entity and Relationship sets using the Entity-Relationship (ER) model. Below is an itemized description of each of the entity type definitions that make up our Financial Aid organization, as well as the relationships amongst these entity types.

ENTITY TYPE DEFINITIONS:

Employee: <u>EmployeeID</u>, <u>SSN</u> Hire Date, Name, Address, Phone Number, Sex.
An **employee** works for a department and their information is typical of any employee. An Employee is distinguished from another employee by the EmployeeID and Social Security Number.

Student: <u>StudentID</u>, <u>SSN,</u> Academic standing, Name, Address, Phone Number, Sex, Income Status.
A **student** applies for financial aid and receives financial aid. Students are distinguished from each other by their StudentID and Social Security Number. Academic standing can be either "good" or "bad", the former meaning they have good grades and GPA, the latter meaning they do not. An income status for a student is used to designate if the student depends on their parents for financial help, or if the student independently supports themselves.

Parent: <u>SSN</u>, Name, Address, Phone Number, Bday, Sex, Status, Income.
A **parent** provides information for their student's financial aid application. Parents are distinguished by their Social Security Number. The parent's Status defines whether a Parent is a Tax-payer or not. Additionally, an Income attribute is included to show how much a Parent makes per year.

Department: <u>DepartmentID</u>, Name,
A **department** is where an Employee works. All Departments are distinguished from one another by their DepartmentID and Access Code. Access Codes pertain to the three departments (Logistics, Outreach, Review Board) in order to separate them by the specific tasks each department performs.

Logistics: <u>Access Code</u>
Logistics is a specialization of Department. The Logistics entity collects Applications in the form of Data and uses that data to store into the organization's internal information. The unique access code of 2 distinguishes Logistics from outreach and Review Board.

Outreach: Access Code
Outreach is a specialization of Department. The Outreach entity communicates with the schools to promote financial aid to students. The unique access code of 4 distinguishes Outreach from Logistics and Review Board.

Review Board: Access Code
Review board is a specialization of Department. The Review Board entity reviews information (all the data collected by Logistics) and checks the annual Budget for available financial aid packages. The unique access code of 6 distinguishes Review Board from Logistics and Outreach.

School: SchoolID**,** Name, Address
A **school** is attended by students and visited by Outreach department. School can be both a High School or College because Financial Aid applications can begin from High School to College. Schools are distinguished from one another by their SchoolID.

Application: AppID, requestedAmount, Status, Date, approved
Application is filled out and submitted by Students. Applications are distinguished by their AppID and include information like the amount a student requested, as well as whether the Application itself was approved or not. The Application also contains a Status, and this is used to identify whether the application is complete or incomplete.

Data: DataID, Description, Date
Data is collected by Logistics and is stored into information. The Description identifies it as coming from a particular Parent or Student. A date attribute is included in case someone in the organization wants to know when certain Data came into the organization, or if there has been a modification on the data. DataIDs are used to distinguish Data records from each other.

Information: SourceID, Date
Information is the collection of all Data and can be accessed by the Review Board to make decisions on Financial Aid packages. Information is distinguished from each other by a SourceID, which identifies the information specific to a certain source (or applicant).

Budget: BudgetID, Amount, Date
The **budget** is the amount available for financial aid awards. It is important to know the budget amount before offering financial aid packages. The Budget entity contains a BudgetID to distinguish Budgets from one another, an Amount containing the monetary value of each budget, and a Date to keep a historical record of every Budget in the history of the organization.

Financial Aid Package: PackageID, Type, Amount
The **financial aid package** is what will be distributed to a student. A financial aid package can be of different Types such as CalGrant A or CalGrant B. PackageID's are used to distinguish financial aid packages from one another, and Amount attribute is used to show the monetary value of the Financial Aid Package.

RELATIONSHIP TYPE DEFINITIONS:

Employee **works for** *Department*; Cardinality: N...1; Participation: Total
The ***works for*** relationship between Employee and Department is to show that all Employees must work for a Department in the organization. Many Employees will work for one Department, and no Employees can work for more than one Department. Start and End Dates are used in ***works for*** to keep track of employment records.

Student **uses info** from *Parent*; Cardinality: N...1; Participation: Partial
The ***uses info*** relationship between Student and Parent is to show that some Students will use the information from their Parents when filling out a financial aid application. The relationship is important because financial aid applications require parent information if the student can provide it. Many Students will use information from a Parent.

Student **attending** *School*; Cardinality: N...M; Participation: Total
The ***attending*** relationship between Student and School is to show that all Students in the Database must be attending a School. This relationship is important in our organization because only Students attending Schools apply for Financial Aid.

Student **fills out** *Application*; Cardinality: 1...1; Participation: Partial
The ***fills out*** relationship between Student and Application is to show that individual Students will fill out their own Financial Aid Application. Because not all students apply for Financial Aid, not all Students are required to fill out an Application. This fact presents a good way for the Organization to keep track of which students ask for aid.

Outreach **communicates with** *School*; Cardinality: 1...N; Participation: Total
The ***communicates with*** relationship between Outreach and School is to show that one specific Department of the organization (Outreach) will be the one to visit Schools to promote the availability of Financial Aid. The Outreach Department's sole purpose is to visit Schools, therefore it will visit many Schools.

Application **becomes** *Data*; Cardinality: N...M; Participation: Partial
The ***becomes*** relationship between Application and Data is to show how within the Organization Applications come in *and then* they become Data. This distinction is very important for the organization and the database because Applications contain a lot of information, but not all Applications actually get submitted. It is only submitted Applications that become Data.

Data **are collected by** *Logistics*; Cardinality: N...1 ; Participation: Total
The ***are collected by*** relationship between Data and Logistics is to demonstrate the primary role of the Logistics Department. Logistics takes in Applications as Data and processes it internally in this way. Because of this all Data must be collected by the Logistics Department.

Logistics **uses** *Data* and **stores into** *Information*; Cardinality: 1...N...M; Participation: Total
The ***uses and stores into*** relationship between Logistics, Data, and Information is to demonstrate another role Logistics plays in the organization. Think of Logistics as employees doing data entry, but in this relationship they are submitting their Data into the Information storage the organization has for executive decision making.

Information **is reviewed by** *Review Board;* Cardinality: N...1; Participation: Total
The ***is reviewed by*** relationship between Information and Review Board is used to show how the Review Board department will make decisions as to who gets Financial Aid. Information is the total collection of internal data that has already been successfully processed, so the Review Board Department looks here when making decisions.

Review Board **checks** *Budget*; Cardinality: 1...1; Participation: Total
The ***checks*** relationship between Review Board and Budget is used to show a very important part of the decision-making process of the Review Board Department. That very important part is to check the annual Budget that is given to the organization to determine how much financial aid will be given.

Budget **allocates funds to** *Financial Aid Package*; Cardinality: 1...N; Participation: Total
The ***allocates funds to*** relationship between Budget and Financial Aid Package is used to show that the Budget is the determining factor of how much money goes into a Financial Aid Package. This is done for every Financial Aid Package.

Financial Aid Package is **distributed to** *Student*; Cardinality: N…M;
Participation: Partial
The ***distributed to*** relationship between Financial Aid Package and Student demonstrates the final aspect of the Organization, and that is Financial Aid disbursement. Many Financial Aid Packages are distributed to many of the Students who applied for Financial Aid and whose Applications were accepted.

1.2 CONCEPTUAL DATABASE DESIGN

The conceptual database design assists in creating an accurate database representation of an organization. This is best achieved by using the Entity-Relationship (ER) model. The ER model creates a representation of our organization's data by using *Entities* and *Relationships*. These Entities and Relationships will make up our Sets. An *Entity Set* is the collection of all entities of a particular entity type in the database at any point in time. Similarly, a *Relationship Set* is a set of relationship instances amongst entities in the database. Earlier in our report we introduced our Entity types and Relationship types briefly, now we will give detailed descriptions of these Entity and Relationship types.

1.2.1 ENTITY SET DESCRIPTION

Entity Name: Employee

Description: The Employee entity stores basic information for each employee such as as age, name and ssn. An entry is created only when a new employee is hired. Entries are never deleted except in extreme circumstances. However, entries may be updated when variances in information occurs (i.e. address change). Additionally, each employee will be a part of a department which will generalize the employee's responsibilities and skill set. Employees are only allowed to work for a single department and, therefore, cannot have multiple roles in the organization. Employees are distinguished from one another by the EmployeeID and Social Security Number.

Candidate Keys: EmployeeID, SSN
Primary Keys: EmployeeID
Entity Type: Strong Entity
Fields to be Indexed: EmployeeID, SSN
Attributes:

Attribute Name	***EmployeeID***	***Name***	***Birthdate***	***Sex***	***Address***
Description	ID distinguishes employees from one another.	First name, Middle name, Last name	Date of Birth	Male or Female	Street Address , City, State, Zip code
Domain / Type	Integer	String, String, String	Date	Character	String, string, string, integer
Value / Range	0 to MaxID	Any, Any, Any	Any	M or F	Any, Any, Any, 00000-99999
Default Value	None	None	None	None	None
Null Value Allowed	No	Yes, Middle Initial only	No	No	No
Unique	Yes	No	No	No	No
Single or Multi-Value	Single	Single	Single	Single	Single
Simple or Composite	Simple	Composite	Simple	Simple	Composite

Employee Attributes (continued)

Attribute Name	**SSN**	**Hire Date**	**Phone Number**
Description	Federal Identification number	Starting date of employment.	Employee primary contact phone number.
Domain / Type	Integer	Date	Integer
Value / Range	000000000 - 999999999 (9 digit integers)	Any	00000000000 - 99999999999 (11 digit integers)
Default Value	None	None	None
Null value allowed	No	No	Yes
Unique	Yes	No	No
Single or Multi-value	single	single	single
Simple or Composite	Simple	Simple	Simple

Entity Name: Student
Description: The student entity contains information of a typical student, such as age and name. A student is created only when a student completes an application. All students are distinguished from each by their Student ID. In an organization such as financial aid, where there can be thousands of student applicants, it is important to distinguish them from one another. Students are distinguished from each other by their StudentID and Social Security Number. Academic standing can be either "good" or "bad", the former meaning they have good grades and GPA, the latter meaning they do not. An income status for a student is used to designate if the student depends on their parents for financial help, or if the student independently supports themselves.

Candidate Keys: StudentID, SSN
Primary Keys: StudentID
Entity Type: Strong Entity
Fields to be Indexed: StudentID, SSN
Attributes:

Attribute Name	**StudentID**	**SSN**	**Academic Standing**	**Name**	**Birthdate**
Description	ID distinguishes students from one another	Federal identification number	student can be in good standing or bad standing	First name, Middle name, Last name	Date of Birth
Domain/Type	Integer	Integer	Character	String, String, String	Date
Value/Range	0 to MaxID	000000000 - 999999999	G or B (good,bad)	Any, Any, Any	Any
Default Value	None	None	G	None	None
Null Value Allowed	No	Yes	No	No	No
Unique	Yes	Yes	No	No	No
Single or Multi-value	Single	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Composite	Simple

Student attributes continued:

Attribute Name	**Address**	**Phone Number**	**Sex**	**Income Status**
Description	Street Address, City, State, Zip code	Student primary contact phone number	Gender	Independent or Dependent financially
Domain/Type	String, String, String, Integer	Integer	Character	String
Value/Range	Any, Any, Any, 00000-99999 (9 digit integers)	00000000000 - 99999999999 (11 digit integers)	M or F	“Independent” or “Dependent”
Default Value	None	None	None	Dependent
Null Value Allowed	No	No	No	No
Unique	No	No	No	No
Single or Multi-valued	Single	Single	Single	Single
Simple or Composite	Composite	Simple	Simple	Simple

Entity Name: Parent
Description: The parent entity contains information a student uses when filling out an application. Financial Aid applications require parent information, such as whether they are taxpayers and how much income they have per year. The parent's Status defines whether a Parent is a Tax-payer or not. Additionally, an Income attribute is included to show how much a Parent makes per year.

Candidate Keys: SSN
Primary Keys: SSN
Entity Type: Strong Entity
Fields to be Indexed: SSN, Name
Attributes:

Attribute Name	**SSN**	**Name**	**Address**
Description	Federal Identification number	First name, Middle name, Last name	Street Address, City, State, Zip code
Domain / Type	Integer	String, String, String	String, String, String, Integer
Value / Range	000000000 - 999999999 (9 digit integers)	Any, Any, Any	Any, Any, Any, 00000-99999
Default Value	None	None	None
Null value allowed	No	Yes, Middle Initial only	No
Unique	Yes	No	No
Single or Multi-Value	Single	Single	Single
Simple or Composite	Simple	Composite	Composite

Parent Attributes continued:

Attribute Name	Birthdate	Phone Number	Sex	Status	Income
Description	Date of Birth	Phone Number	Male or Female	Taxpayer or Non-Taxpayer	Yearly Income
Domain / Type	Date	Integer	Character	String	Double
Value / Range	Any	00000000000 - 99999999999 (11 digit integers)	M or F	"Taxpayer" or "Non-Taxpayer"	0.00 to 999,999.00
Default Value	None	None	None	None	None
Null value allowed	No	No	No	Yes	No
Unique	No	No	No	No	No
Single or Multi-Value	Single	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple	Simple

Entity Name: Department
Description: The department entity is a superclass on the three different departments of our organization. Those three department subclasses are: Logistics, Outreach, and Review Board. The intention behind creating the Department entity as a superclass is due to the fact that the subclasses are only distinguished by what they can or cannot do. For example, Logistics Department would handle data entry, but Outreach only visits schools and gets students to fill out applications. All employees must work for a department, and every department is involved in a specific task. In our organization, having this separation ensures a streamlined workflow amongst teams.

Candidate Keys: DepartmentID, Name
Primary Keys: DepartmentID
Entity Type: Strong Entity
Fields to be Indexed: DepartmentID, Name
Attributes:

Attribute Name	**DepartmentID**	**Name**	**Access Code**
Description	ID distinguishing Departments from one another	Name given to specific department	Access Code granting certain abilities for a department.
Domain / Type	Integer	String	Integer
Value / Range	3 (logistics), 5 (Outreach), 6 (Review Board)	"Logistics", "Outreach", "Review Board"	2 (logistics), 4 (Outreach), 6 (Review Board)
Default Value	None	None	None
Null value allowed	No	No	No
Unique	Yes	Yes	Yes
Single or Multi-Value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Logistics
Description: Logistics is a department of the financial aid organization. The purpose of Logistics department is to collect Applications in the form of Data and to use that Data to store into the Information entity which keeps track of all Student application information. Logistics has a special access code value which allows the department to accomplish it's tasks. No other department will have the same access code as Logistics, which also makes the department 100% responsible for all data entry and data collection.

Candidate Keys: Access Code
Primary Keys: None
Entity Type: Weak Entity
Fields to be Indexed: Access Code, DepartmentID(from superclass)
Attributes:

Attribute Name	**Access Code**
Description	Code granting special responsibility to Department.
Domain / Type	Integer Constant
Value / Range	2
Default Value	2
Null Value Allowed	No
Unique	Yes
Single or Multi-Value	Single
Simple or Composite	Simple

Entity Name: Outreach
Description: Outreach is a department of the financial aid organization. The purpose of the outreach department is to visit schools to inform students about financial aid services, as well as to get students to sign financial aid applications. Outreach is a very important aspect of the financial aid service because it exposes those who are unaware of financial aid to the option. Outreach is also a subclass of Department. Department will hold basic information regarding a general Department, but Outreach will contain a specific access code, granting them the ability to visit schools and gather student applications.

Candidate Keys: Access Code
Primary Keys: none
Entity Type: Weak Entity
Fields to be Indexed: Access Code, DepartmentID (from superclass)
Attributes:

Attribute Name	Access Code
Description	Code granting special responsibility to department
Domain / Type	Integer Constant
Value / Range	4
Default Value	4
Null Value Allowed	No
Unique	Yes
Single or Multi-value	Single
Simple or Composite	Simple

Entity Name: Review Board
Description: From a student perspective, the Review Board could be considered the most important department in the financial aid organization. The reason is that Review Board has the special task of determining who will get financial aid. The Review Board works closely with the Information provided by the Logistics Department, as well as with a yearly Budget granting a certain amount of funds for the final financial aid package. Like the other subclasses of Department, the access code grants Review Board department the ability to do its specific tasks without any interference and 100% responsibility.

Candidate Keys: Access Code
Primary Keys: None
Entity Type: Weak Entity
Fields to be Indexed: Access Code, DepartmentID (from superclass)
Attributes:

Attribute Name	**Access Code**
Description	Code granting special responsibility to Department
Domain / Type	Integer Constant
Value / Range	6
Default Value	6
Null Value Allowed	No
Unique	Yes
Single or Multi-Value	Single
Simple or Composite	Simple

Entity Name: School
Description: School is attended by a Student and communicates with Outreach. The school entity contains basic information about a school, such as the name and address. A schoolID is used to separate data between different schools. The ID is necessary for situations where more than one school has the same name. In our database design the School can represent a High School or a College.

Candidate Keys: SchoolID
Primary Keys: SchoolID
Entity Type: Strong Entity
Fields to be Indexed: SchoolID
Attributes:

Attribute Name	**SchoolID**	**Name**	**Address**
Description	ID distinguishes schools from one another	Name of School	Street Address, City, State, Zip code
Domain / Type	Integer	String	String, String, String, Integer
Value / Range	0 - MaxID	Any	Any, Any, Any, 00000-99999
Default Value	None	None	None
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-Value	Single	Single	Single
Simple or Composite	Simple	Simple	Composite

Entity Name: Application
Description: An application is filled out by a student to quality for financial aid. Applications are given to students by Outreach, but generally a student can acquire an application through other means. Applications can have a status of complete or incomplete. These statuses are useful for the Logistics department when they collect applications as data. In a real-world setting, Application statuses can change throughout the application process. The Application entity will also contain the amount a student requests when filling out an application, as well as whether an application was approved or not. The Application Entity will contain an identification number to distinguish it from other applications, a date corresponding to the initial submission of the application, a status message, and a requested amount.

Candidate Keys: AppID
Primary Keys: AppID
Entity Type: Strong Entity
Fields to be Indexed: AppID

Attribute Name	AppID	requestedAmount	Date	Status
Description	AppID distinguishes applications from one another.	Student requests amount they desire.	Date the application was submitted.	Status to know what was completed in application.
Domain / Type	Integer	Double	Date / Time	String
Value / Range	0 - MAX	0.00 to 99,999.00	Any	“Complete”, “Incomplete”,
Default Value	None	None	None	None
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

Application Attributes continued:

Attribute Name	approved
Description	Designates the approval status of application
Domain / Type	Boolean
Value / Range	True or False
Default Value	None
Null Value Allowed	Yes
Unique	No
Single or Multi-value	Single
Simple or Composite	Simple

Entity Name: Data
Description: Data is the total collection of all applications in the organization. We make the distinction between Applications and Data because Applications contain Data. This Data can be from a parent or a student, but it comes from the same application. Data is distinguished by DataID's. Dates on Data are used to identify when Data has been accessed or modified. Considering all of this as Data is beneficial to the Logistics department as they collect Data for entry into their system.

Candidate Keys: DataID
Primary Keys: DataID
Entity Type: Strong entity
Fields to be Indexed: DataID, Description, Date
Attributes:

Attribute Name	DataID	Description	Date
Description	DataID is how to distinguish Data from one another.	Distinguishes between parent data and student data.	When data is used, Date defines when it was accessed.
Domain / Type	Integer	String	Date / Time
Value / Range	0 - MAX	"Parent", "Student"	Any
Default Value	No	No	No
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-Value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Information
Description: Information represents the total collection of data the organization has. Logistics Department collected Data from Applications, but that Data gets stored into the Information entity. Information is distinguished from each other by a SourceID, which is an internal way to identify the source of information. This entity is crucial for the Review Board, because it is where they see all applicant data, current and past. A good analogy to the Information entity could be a file cabinet, where each file has a distinguished identification, a date, and description associated with it.

Candidate Keys: SourceID
Primary Keys: SourceID
Entity Type:
Fields to be Indexed:
Attributes: SourceID, Date

Attribute Name	SourceID	Date
Description	Number that uniquely identifies specific information of an applicant internally.	Date information was created.
Domain / Type	integer	Day/ Time
Value / Range	0 - MAX	Any
Default Value	None	None
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-Value	Single	Single
Simple or Composite	Simple	Simple

Entity Name: Budget
Description: The Budget entity stores information about a Financial Aid budget. A financial aid budget is the possible amount that can be allocated to all students every year. Because a financial aid budget is never the same, an ID is given to distinguish past budgets from a current budget. A budget amount is also included in the entity as the monetary value is necessary to create an appropriate financial aid package. A Date attribute is included to keep a historical record of every Budget in the history of the organization. The Budget is checked by the Review Board when determining who gets financial aid.

Candidate Keys: BudgetID
Primary Keys: BudgetID
Entity Type: Strong Entity
Fields to be Indexed: BudgetID
Attributes: BudgetID, Amount, Date

Attribute Name	**BudgetID**	**Amount**	**Date**
Description	ID distinguishes a current budget from previous budgets	Monetary value that can be allocated for financial aid.	Date of yearly budgets for historical purposes.
Domain / Type	Integer	Double	Day/Time
Value / Range	0 to MaxID	0.00-999,999.99	Any
Default Value	None	0.00	None
Null Value Allowed	No	No	No
Unique	Yes	No	Yes
Single or Multi-Value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Financial Aid Package
Description: After the entire financial aid process is completed, from student application submissions to review board decisions, students receive a letter of their Financial Aid Package. This typically includes an estimated amount the student will receive, as well as the different types of awards they will receive (Cal Grant, Pell Grant, for example). A PackageID is used to specify unique packages throughout the lifetime of the organization, and a Type can be associated with an ID to look for a particular award to a student.

Candidate Keys: PackageID
Primary Keys: PackageID
Entity Type: Strong Entity
Fields to be Indexed: PackageID, Type
Attributes: PackageID, Type, Amount

Attribute Name	**PackageID**	**Type**	**Amount**
Description	ID distinguishes financial aid packages from one another.	Type designates the possible financial aid awards granted to a student.	Budget as monetary value.
Domain/ Type	Integer	String	Double
Value / Range	0 to MaxID	"CalGrantA", "CalGrantB", "Other"	0.00 - 999,999,999.00
Default Value	None	None	0.00
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-Value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

1.2.2 Relationship Set Description

A relationship is an association among entity types. A relationship set is the total relationships among entities in a Database. Some relationships also contain attributes in order to explain the relationship clearly. Relationships also specify constraints on how many entities are related to each other, and how many entities must participate in a relationship. Below we will define each relationship type with the entity types it relates, the constraints on cardinality and participation, as well as any attributes adding detail to the relationships.

Relationship: Works For
Description: All Employees are hired for a specific department. They will work in particular tasks only and cannot cross over into other Departments. Because of this the cardinality must be N...1. All employees must work for a Department.
Entity Sets Involved: Employee, Department
Mapping Cardinality: N...1
Descriptive Field: Start Date, End Date
Participation Constraint: Total

Relationship: Uses Info
Description: When a student fills out an application, there are pages of questions the Student must answer such as their Academic standing and if they are financially dependent on their parents. Sections of the application are generally split into a Student side and Parent side. The Student entity has all the necessary information for their own information, but a Parent entity must be used containing all the information needed to finish the application. Not every student is able to provide this info due to circumstances where a student lost a parent or doesn't know their real parents.
Entity Sets Involved: Student, Parent
Mapping Cardinality: N...1
Descriptive Field: None
Participation Constraint: Partial

Relationship: Attending
Description: The Attending relationship emcompasses a student attending a high school or a student planning on attending a college. This is important because only a High School or College student can apply for financial aid. Many students will attend many different schools, and all students must attend a school.
Entity Sets Involved: Student, School
Mapping Cardinality: N...M
Descriptive Field: Start Date, End Date
Participation Constraint: Total

Relationship: Fills Out
Description: A student fills out an application for financial aid every year. All students applying for financial aid must fill out one application. Because of this, not all students participate in the relationship because some students will forget to apply for financial aid, or simply not pursue a financial aid opportunity.
Entity Sets Involved: Student, Application
Mapping Cardinality: 1...1
Descriptive Field: Date
Participation Constraint: Partial

Relationship: Communicates With
Description: It is the primary job of the Outreach department to communicate with schools. Because of this, the entity department will find itself visiting a large number of schools (in our organization all throughout california) in order to accomplish their job requirements. The Outreach department has no exception to this rule, so their participation is total.
Entity Sets Involved: Outreach, School.
Mapping Cardinality: 1...N
Descriptive Field: Date
Participation Constraint: Total

Relationship: Becomes
Description: At first, an Application is only a form. The form is either on paper or electronic. Once this application form has been submitted, it is converted to Data. This distinction has to happen because not every application will be turned in, so only those applications that are submitted will be collected by a specific department as Data.
Entity Sets Involved: Application, Data
Mapping Cardinality: N...M
Descriptive Field: None
Participation Constraint: Partial

Relationship: Are Collected By
Description: Once an Application has become Data, the Logistics department will Collect that Data for internal use. Data encompasses all submitted applications, so all Data goes to one Logistics department. Because Data is the collection of all submitted Applications, the participation is Total.
Entity Sets Involved: Data, Logistics
Mapping Cardinality: N...1
Descriptive Field: Date/Time
Participation Constraint: Total

Relationship: Uses ___ and Stores Into ___
Description: Logistics now has collected all Data, and their next step is to Use Data and Store into Information. Information encompasses all of the Data that has been internally processed. This implies that all information can now be accessed by the Review Board for financial aid awards and any other decision making necessary to the organization. Only the Logistics department uses all of the Data to store into Information, implying a total participation.
Entity Sets Involved: Logistics, Data, Information
Mapping Cardinality: 1…N…M
Descriptive Field: None
Participation Constraint: Total

Relationship: Is Reviewed By
Description: Once Data is stored as Information, Information is reviewed by the Review Board department in order to make executive decisions on what financial aid packages will be distributed and to whom they will go. All of the information will be reviewed by one Review Board department. All of the information is needed in order for this to happen, so Total participation is required from Information to Review Board.
Entity Sets Involved: Information, Review Board
Mapping Cardinality: N…1
Descriptive Field: Date / Time
Participation Constraint: Total

Relationship: Checks
Description: The Review Board department must first Check their Budget before they offer financial aid packages. This ensures that they do not promise amounts of money they do not have. Only one department looks at one current Budget, as the Budget will change every year.
Entity Sets Involved: Review Board, Budget
Mapping Cardinality: 1…1
Descriptive Field: None
Participation Constraint: Total

Relationship: Allocates Funds To
Description: The Budget determines the funds that will be used for Financial Aid Packages. One Budget is used for the entirety of Financial Aid Packages, and the Budget must always be used to Allocate these funds.
Entity Sets Involved: Budget, Financial Aid Package
Mapping Cardinality: 1…N
Descriptive Field: None
Participation Constraint: Total

Relationship: Distributed To
Description: The final step of our organization is to Distribute Financial Aid Packages to Students. Because this process was started with an application, it is understood that not every student will receive a Financial Aid Package. This is usually due to bad academic standing or to a parent having too high of an income. Regardless, many Financial Aid Packages will be Distributed To many Students.
Entity Sets Involved: Financial Aid Package, Student
Mapping Cardinality: N…M
Descriptive Field: Date / Time
Participation Constraint: Partial

1.2.3 RELATED ENTITY SET

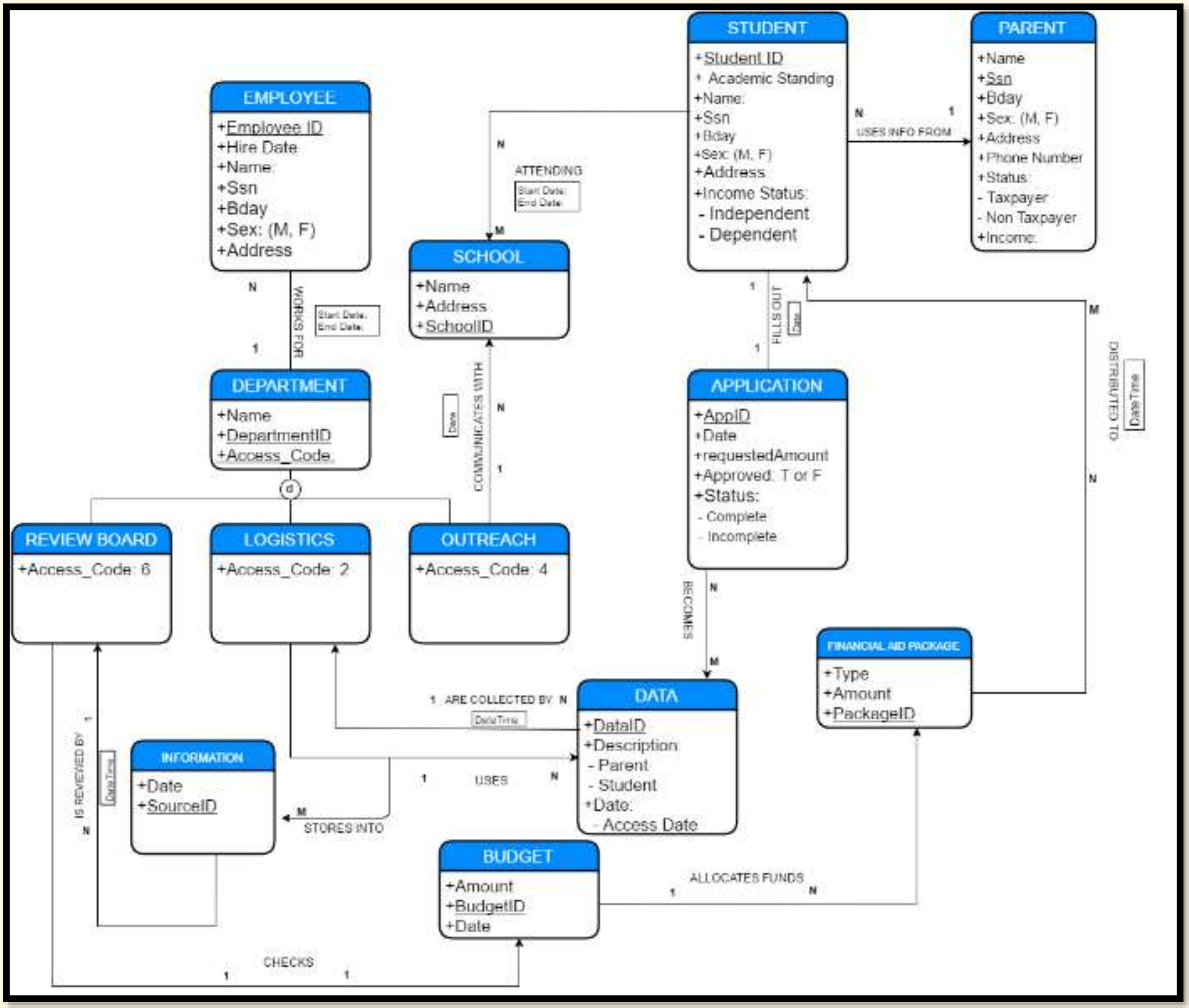
Specialization is the process of defining a set of subclasses of an entity type. When using Specialization, the Entity is known as the Superclass of the Specialization. The set of subclasses are defined on the basis of a distinguishing characteristic of the entities in the superclass. For our organization, we used Specialization on the Department Entity to distinguish the different tasks each Department is involved in.

Generalization is the process bringing multiple related subclasses together under a superclass. Our organization Database created a Generalization of the Logistics, Outreach and Review Board subclasses into the Department superclass because all of the information is shared except for unique access codes determining what each subclass can or cannot do.

Both Specializations and Generalizations have constraints (completeness, disjointedness). Our database Specialization of Department to Logistics, Outreach and Review Board are *disjoint* because an Employee can only work for one particular department. A Department cannot be two departments at once, different departments must be unique from each other. On the other hand, the completeness constraint of our specialization is *total* because a Department must be one of the three possible departments (Logistics, Outreach, Review Board). This is necessary to prevent the introduction of irrelevant departments to our organization.

1.2.4 ER DIAGRAM

The Entity-Relationship Diagram (ER Diagram) is important to have as a visual representation of all the entities and relationships of an organization's Database. In an ER Diagram, all of the Entities, Relationships, Cardinalities, and Participation Constraints that were described throughout this report are drawn. Below is our organization's ER Diagram.



[This page intentionally left blank]

Phase Two:
Conceptual Database and Logical Database

We previously created a conceptual database by making use of the E-R model. The advantage of the E-R model and conceptual database design is that it makes it easy to visualize an organization's database representation. On the other hand, a logical database is used in order to have a clear representation of how a database will be stored in a software implementation. In order to move from the previous conceptual design to a logical design, a conversion from E-R model to the Relational model is required.

Section 2.1 describes the E-R model and the relational model. We will describe the history of each model, purposes, as well as similarities and differences.

Section 2.2 will be a detailed description of how to convert a conceptual database model to a relational model. This section will include the methods required for converting entity types and relationships to relations.

Section 3 will consist of an implementation of the relational model on our previous conceptual model. Once the relational model has been created, we will demonstrate a relational database of our organization with sample data.

Section 4 will consist of queries we have created for our relational database. These queries will be created using three formal querying languages: relational algebra, tuple relational calculus, and domain relational calculus.

2.1 E-R Model and Relational Model

The E-R model is used to represent a conceptual database design. Now that we have created a conceptual database with the E-R model, a logical database follows. The logical database will require the use of the Relational model. These models will be described and compared below.

2.1.1 Description of E-R Model and Relational Model

The E-R model and Relational model have a lot of differences in their purpose and features. The E-R model is a high-level conceptual data model frequently used for the conceptual design of a database. The purpose of the E-R model is to create an organization's database with the use of simple objects and the relationships amongst those objects. Using the E-R model makes the conceptual design of a database simple enough for users, business owners, and designers to understand.

The Relational model was first introduced by Ted Codd of IBM Research in 1970. The relational model was popular due to its simplicity and mathematical foundations via the use of mathematical relation, as well as set theory and first-order predicate logic. The relational model is a method for creating a logical database design, meaning that it demonstrates how the database will be implemented in a software application. The relational model has been implemented in a large number of commercial systems, such as Oracle DBMs, SQLServer, and MySQL. Whereas the E-R model presents data as objects and their relationships to one another, the relational model represents data as "relations" only.

The E-R model and Relational model have different major features. The E-R model makes use of "entities", which represent the objects that make up the organization, and "relationships", which describe how different entities are associated with each other. The best way to represent these features is with a diagram in which entities are boxes and relationships are lines connecting the boxes to one another. Entities and relationships both contain attributes describing information each entity or relationship contains, and these attributes can be grouped together so they are easier to understand. Additionally, relationships have constraints to show how many times entities can be related to each other.

The Relational model is defined by the concept of a "relation". The relational model consists of "tuples," which are single, flat lists of related values. Relational schema are lists of attributes that describe the purpose of each value in a tuple, as well as the domain of those values. A Relational instance consists of all the tuples that belong to the same relational schema. This can be pictured as a table where columns contain attributes and rows contain individual tuples.

The core difference between the E-R model and Relational model is their purpose. The E-R model presents an abstract visualization of an organization's database so that it is easy to understand amongst business owners and those who will design the database. This makes for better communication during database design. On the other hand, the Relational model is used to represent the database as it will actually be implemented in a software application. For this reason, the relational model is not as simple as the E-R model, and it involves the use of mathematics and mathematical querying languages.

2.1.2 Comparison of Two Different Models

There are many advantages to using the E-R model over the Relational model. The E-R model is better for communicating ideas between those who implement the database software and those in the business that are not capable of understanding all the technicalities of software engineering. This is important because the majority of software development happens in the professional world. The E-R model accomplishes this flexibility by excluding the details of software implementation. The E-R model focuses more on the overall conceptual design of the Database.

The E-R model also has many disadvantages. Because the E-R model is geared toward creating an understanding amongst non-engineers, this means it has to lack a lot of the logical qualities used in the software implementation. For example, because E-R doesn't implement any type of mathematical logic, it is impossible to create Queries with a mathematical Query Language purely from the E-R model. Additionally, there are many ways to design an E-R model, so there are multiple ways to create an E-R diagram for the same database.

The Relational model also has advantages and disadvantages. The relational model's advantages are that it is formal and standardized, as well as a good tool for creating the software implementation of the database. The Relational model is formal and standardized in its use of Discrete mathematics and formal querying languages like the Relational Algebra and Relational Calculus. The Relational model is also a good tool for software implementation because of its features like relation schema and relation instances. These features can be easily translated into computer data.

The Relational model's disadvantages are mainly due to its technical nature. For example, in a Relational model Entities and Relationships between Entities are reduced to Relations. This is a disadvantage for someone with a non-technical background, because it doesn't paint a clear picture of the Database like the E-R model does. Only someone that has a deep understanding of the Relational Model can actually make use of it, resulting in a lack of use for the Organization in general.

The E-R model and Relational model have many similarities. For example, both models can represent the structure of data in a clear way. The E-R model represents data by using Entity and Relationship types, whereas the Relational model represents data by using Relations. The E-R Entities and Relationships contain names and lists of attributes. The Relational Relations also contain names and list of attributes. Both models make use constraints, which define how the data will be related to one another. Additionally, both the E-R and Relational models use the idea of Tuples (or instances in E-R) and Schema (or types in E-R).

The E-R and Relational model also have differences. The differences are mainly due to the fact that the Relational model describes everything through the concept of a Relation, whereas the E-R model describes everything through the concepts of Entities and Relationships. Additionally, The Relational model does not allow for the use of composite, multivalued, attributes. This is not the case for the E-R model, which allows for the use of composite and multivalued attributes. The Relational model is different from the E-R model in that the Relational model has no cardinality and participation constraints, but instead makes use of integrity constraints to create consistency between relations that reference each other.

2.2 Converting from Conceptual Database to Logical Database Model

Previously we described the E-R and Relational models by noting their advantages, disadvantages, similarities, and differences. Now, we will give an in-depth analysis of the conversion process from E-R to Relational model. First the conversion of Entity types to Relations will be described, then Relationship types to Relations, and finally we will describe building constraints to ensure the Relational model data has integrity.

2.2.1 Converting Entity Types to Relations

Conversion of E-R Model to Relational Model requires that all entity types be represented as a set Relation Schemas. A Relation Schema contains a list of attributes with single-value domains. This differs from the ER Model entity types which contain multi-value attributes, as well as weak entities with no key.

The following section will explain how to convert both Strong Entities and Weak Entities into Relations. Then we will explain how to properly map all Simple and Composite attributes as Relation attributes with atomic domains. Afterword, we will explain how to properly map all single and Multi-Value attributes into Relation attributes.

Converting Strong Entities into Relations

When converting a Strong Entity into a Relation, every strong entity type E will be converted into a Relation Schema R. The relation schema has the same name as the strong entity. The attributes of the Relation Schema will be the simple, single-valued attributes of the Entity Type, as well as the simple components of the Entity Type's composite attributes. The next step is to choose one of the Key attributes of E as the Primary Key for R. If the chosen key of E happens to be a composite attribute, then the set of simple attributes that form it will together form the primary key of R. Any additional key attributes of the Entity Type E will become candidate keys of the Relation Schema R.

Converting Weak Entities into Relations

When converting a Weak Entity into a Relation, every weak entity type W will be converted into a Relation Schema R, which includes all the simple attributes of W, as well as simple components of *W*'s composite attributes. *R* has the same name as the Weak Entity *W*. In order to properly take care of mapping the Weak Entity *W* to *R*, the primary key of the relation for *W*'s owner entity becomes a foreign key in *R*. The combination of the foreign key and all of *W*'s partial keys make up the Primary Key of the new Relation *R*.

Mapping Simple and Composite Attributes

When mapping Simple and Composite attributes of an Entity Type, the Simple attributes of an Entity Type *E* become attributes of the corresponding Relation Schema *R*. If the Entity Type *E* contains any composite attributes, then the simple components of those composite attributes each become individual attributes in the corresponding Relation Schema *R*.

Mapping of Single and Multi-Value Attributes

When mapping Single or Multi-Value attributes of an Entity Type *E*, there are multiple scenarios that must be handled. First, all of the Single-Valued attributes of an Entity Type *E* will become attributes of the Relation Schema *R*. The challenging scenario is when The Entity Type *E* contains multi-valued attributes. If multi-valued attributes exist, then for each multivalued attribute *A*, a Relation Schema R_A must be created that includes the attributed corresponding to *A* plus the primary key *K* of Entity *E* as a foreign key of R_A. The Primary key of R_A is the combination of *A* and *K*. If the multivalued attribute is composite, then we include the simple components of the composite attribute.

2.2.2 Converting Relationship Types to Relations

In the previous section we converted Entities to Relations by following a certain set of rules. Now we will convert Relationship Types following a certain set of rules. The relational model only applies the concept of a Relation, which means that all Relationship Types from the E-R model must now be represented as Relation Schemas. Converting Relationship Types to Relations is not the same as with Entities, there are certain rules that need to be followed, as well as certain specifics of Relationships that guide how the conversion will occur. In this section, we will explain how the methods for converting the following Relationships:

- Cardinality constraints (one-to-one, one-to-many, many-to-one, many-to-many)
- Superclass/Subclass concepts "IsA" and "HasA"
- Relationship types involving other Relationship Types
- Recursive Relationships (involving one entity type)
- Relationships with more than 2 Entity Types
- Relationship and Union Types (Categories)

Mapping of Binary Relationship types with a 1:1 Cardinality Constraint

For each binary 1:1 Relationship Type *R*, identify the Relations *A* and *B* that correspond to the Entity Types participating in *R*. Because it is 1:1, this means that each instance of Relation *A* should be related to exactly one instance of Relation *B.* There are three possible approaches to handle this constraint.

1. ***Foreign Key Approach***: In this approach, the primary key of Relation *A* is made into the foreign-key attribute of Relation *B* (or vice versa). All simple attributes and simple components of composite attributes that belonged to the Relationship Type *R* also become

   attributes of the Relation containing the Foreign Key. The Foreign Key Approach is very good because it decreases the number of join operations when doing a query. The Foreign Key Approach should only be used if one of the Entity Types has a *Total* participation in the Relationship, otherwise the Foreign Key will be NULL for relations that do not participate, and this will be a waste of data storage.

2. ***Merged Relation Approach:*** In this approach, the attributes of Relation *A* and Relation *B* are combined into a single Relation *S*. The Merged Relation Approach is not very good. This is because if two relations can be combined into one, then their E-R Entity Types should have been combined during the Conceptual design phase.

3. ***Cross-reference or Relationship Relation Approach:*** In this approach, a new Relation *S* is created to represent the Relationship Type *R. S* is considered a *Relationship Relation*. *R* will contain the Primary Keys of Relation *A* and Relation *B* as Foreign Key attributes. Simple attributes and simple components of composite attributes will be included in the new Relation *S*. The Primary Key of Relation *S* is one of the Foreign Keys.

   The Cross-reference or Relationship Relation Approach is very good when both of the participating entity types do not have total participation. This increases the number of Joins in Queries.

Mapping of Binary Relationship types with a 1:N Cardinality Constraint

For each binary 1:N Relationship Type *R*, identify the Relations *A* and *B* that correspond to the Entity Types participating in *R*. Because it is 1:N, this means that each instance of Relation *A* could be related to many instances of Relation *B*, and each instance of Relation *B* can be related to only one instance of Relation *A*. There are two approaches to handling this constraint.

1. ***Foreign Key Approach:*** The Foreign Key approach is the same as for Binary 1:1 Relationship Types. The difference lies in that the Foreign Key and Relationship Type attributes must belong to the Relation derived from the entity on the *N*-side of the Relationship (Relation *B*). The reason for this is that

entities on the *N*-side can only be related to at most one entity on the *1*-side of the Relationship Type.

2. ***Cross-reference or Relationship Relation Approach:*** The Cross-reference approach is the same as for Binary 1:1 Relationship Types, except the primary key of the new Relation *S* must be the Foreign Key of the Relation on the *N*-side of the relationship.

The advantages and disadvantages for these approaches in a Binary 1:N Relationship are the same as for a Binary 1:1 Relationship.

<u>Mapping of Binary Relationship types with a M:N Cardinality Constraint</u>

For each binary M:N Relationship Type *R*, a new Relation *S* is created to represent *R*. Because the Relationship is M:N, this means that each instance of Relation *A* can be related to multiple instances of Relation *B*, and each instance of Relation *B* can also be related to multiple instances of Relation *A*.

1. ***Cross-reference or Relationship Relation Approach:*** With a Binary M:N Relationship, the only method for converting the Relationship Type is through the Cross-reference approach. The Primary Keys of *both* the Relations that represent the participating Entity Types (*A* and *B*) form the Primary Key of *S*. All simple attributes of the Relationship types as well as simple components of composite attributes are included in the new Relation *S*.

<u>Mapping of Superclasses and Subclasses for the "IsA" Relationship</u>

The "IsA" Relationship refers to the *disjoint* Subclasses of a Superclass Entity Type. This means that the Entity belongs to only one Subclass. There are three possible approaches for representing this Relationship with Relations.

1. ***Multiple Relations - Superclass and Subclass:*** In this approach, a Relation *S* is created for the Superclass *C*, and a Relation *L* is created for each Subclass of *S*. The Superclass Relation *S* contains the attributes of the Superclass Entity *C*. The Subclass Relation *L* contains the attributes of the Subclass Entity, as well as the Primary Key of the SuperClass Relation *S* as a Foreign Key attribute, but it acts as a Primary Key in the Subclass Relation *L*.

The advantages of this approach are that it works for every Superclass Relationship, but it requires more Join operations during Queries because of its separate Superclass Relation.

2. ***Multiple Relations - Subclass only:*** In this approach, only those Entities that are Subclasses are given their own Relations. The Subclass Relation *L* contains the Union *U* of the attributes from the Superclass Entity Type *C* and the Subclass Entity Type $C_{subclass}$. This approach only works for a specialization in which the subclasses are *total*, meaning that every Entity in the Superclass must belong to at least one of the Subclasses. This approach is only recommended if the

Specialization has the *disjointedness constraint* ("IsA"). If the specialization is *overlapping*, the same entity may be duplicated in several Relations.

The advantage of this approach is that it requires fewer Join operations during Queries, but it only works for the Relationships with total participation, that is, when an Entity has to belong to one of the subclasses.

3. ***Single Relation with one type attribute:*** In this approach, a single Relation L is created containing the Union of the Attributes from the Superclass C and the attributes from all the Subclasses L_i *combined.* The Relation L also contains a *Discriminating* attribute ("Type") whose value indicates the Subclass L_i to which each tuple belongs, if any.

The advantage of this approach is that it requires fewer Join operations during Queries than all the others. The disadvantages are in that the Relation can become very large due to all the attributes. Another disadvantage is that the attributes of the Subclasses to which an Entity does not belong will be NULL, resulting in a lot of NULL values of the Subclasses do not have similar attributes. That is why this approach is best when the Subclasses are *disjoint* ("IsA") and have few unique attributes.

Mapping of Superclasses and Subclasses for the "HasA" Relationship

The "HasA" Relationship occurs when Entity Types are *Overlapping* Subclasses of a Superclass Entity Type. This means that an Entity can belong to multiple Subclasses. There are two methods for representing these Relationships with Relations.

1. ***Multiple Relations - Superclass and Subclass:*** This approach is the same for the "HasA" Relationship as it is for the "IsA" Superclass/Subclass Relationship. The advantages and disadvantages of the "IsA" Superclass/Subclass Relationship also apply here.

2. ***Single Relation with Multiple Type Attributes:*** In this approach a single Relation L is created with the Union U attributes from both the Superclass C and all of the Subclasses L_i combined. Additionally, a boolean attribute is included for each of the Subclasses L_i. The boolean is used to indicate that a Relation Instance Tuple belongs to the Subclass L_i for which the boolean is True.

The advantage of this approach are that it requires less Joins on Queries. The disadvantage of this approach is that a lot of NULLs will exist because the attributes for the Subclasses will not always be filled in if the Entity does not belong to the Subclass. Having NULLS results in a waste of data.

Mapping of Relationship types involving other Relationship types

In order to map a Relationship Type R_1 that involve another Relationship Type R_2, a Primary Key Attribute must be created for the Relationship Type R_1. Then, the Relationship between the Relationship Types R_1 and R_2 can be mapped with the ***Cross-reference Approach*** or the ***Foreign Key Approach***. Choosing the approach depends on the Cardinality of the Relationship. Because both approaches require a Foreign Key, then the Primary Key of R_1 will be used as the Foreign Key for either approach.

Mapping of Recursive Relationships

A Recursive Relationship is when an Entity Type, converted to a Relation *R*, is related to itself. Two approaches can be taken when mapping this Relationship.

1. ***Foreign Key Approach:*** Using the Foreign Key Approach for a Recursive Relationship requires that the Foreign Key attribute of *R* references the Primary Key of *R*.

   The advantage of the Foreign Key Approach is that it requires less Join operations, but the disadvantage is that Relation instance tuples that do not participate in the Relationship will have NULL values for the Foreign Key, wasting data.

2. ***Cross-reference or Relationship Relation Approach:*** Using the Cross-reference approach for a Recursive Relationship requires the creation of a new Relation R_1 that represents the Recursive Relationship. Two Foreign Keys are used that both reference the Primary Key of *R*, and their combination forms the Primary Key of R_1.

   The advantage of the Cross-reference Approach is that there will not be the problem of having Relation instance tuples with NULL values, but the disadvantage is that more Join operations will be required when using Queries.

Mapping of Relationships with more than 2 Entity Types

When a Relationship Type associates more than two Entity Types, a Relation *R* is created to represent the Relationship Type. *R* will contain the Primary Keys of the participating Entity Types (which are converted to Relations as well) as Foreign Keys. These Foreign Keys are combined to form the Primary Key of *R*, but a Foreign Key could be excluded from the Primary Key if the corresponding Relation represents an Entity on the *1-Side* of a *1:N* Cardinality Constraint.

Mapping of Union Types (Categories)

Union Types (also known as Categories) occur when a Relation for a Subclass Entity belongs to multiple defining Superclass Entities. Because these multiple Superclass Entities have different keys, a new Key attributes known as the ***Surrogate Key*** is created to correspond to the Category. This is because none of the Superclass Entity Keys can be used to exclusively define all entities in the category. The ***Surrogate Key*** is given to the Superclass Entities, and if all these Superclass Entities define the same Subclass, then the values of the ***Surrogate Key*** will be the same for all those Superclass Entities.

2.2.3 DATABASE CONSTRAINTS

In a Database Management System (DBMS), constraints exist in order to ensure that all data has meaning and makes sense for the Database. In order for the Relational model to accurately represent E-R model Entities, certain rules and conditions must be satisfied. If a rule is violated, the data inside the Relational database will not make any sense. This section will cover different types of Constraints and how they are enforced in a DBMS. The
following Constraints will be discussed:

- Domain Constraints
- Entity Constraint
- Primary Key and Unique Key Constraint
- Referential Constraint
- Check Constraints and Business Rules

Domain Constraints
Domain Constraints exist to ensure that the values for each Tuple in a Relation State are within the Domains of their corresponding attributes in the Relation Schema. Examples of Domain Constraints are restricting the value of an attribute to a specific Data Type (such as integer, float, etc.), and to a subset of Values within that Data Type. This constraint is enforced by the DBMS when an attempt is made to change a Tuple to an invalid value via INSERT or UPDATE. The DBMS will reject these changes, NULL them or assign them a default.

Entity Constraints
Entity Constraints exist to ensure that all Tuples belonging to a Relation State have a Primary Key that is not NULL. This Constraint is used in order to uniquely identify each Tuple. Because the Primary Keys cannot be NULL, the DBMS should reject any INSERT and UPDATE operation when a Primary Key attribute is NULL.

Primary Key and Unique Key Constraints
The Primary Key constraint exists to ensure that no two Tuples of the same Relation State have the same values for Primary Key attributes. The uniqueness constraint can exist even if the attribute is not a Primary Key. All of this is used to ensure that Tuples in a Relation State are uniquely identifiable. The DBMS enforces this constraint by not allowing INSERT or UPDATE operations when the values of the Unique Key or Primary Key match an existing Tuple in the Relation State. The DBMS can also auto-increment the Primary Key of each new Tuple during INSERT operations if the attribute has an integer domain.

Referential Constraints
The Referential Integrity Constraint ensures that if a Relation R_a contains a Foreign Key that references Relation R_b, and a Tuple T_a exists in a Relation State of R_a, then the Foreign key of T_a matches the Primary Key for a Tuple T_b that exists in Relation State of R_b.

The DBMS enforces Referential Constraints for INSERT, UPDATE, and DELETE operations.

With INSERT operation, any new Tuple that has an invalid Foreign Key value is rejected, or the value is set to NULL or a default value if possible.

Three options exist for the DELETE operation:

1. ***Restrict:*** When deleting a Tuple that is being referenced by Foreign Keys from other Tuples in the Database, Reject the DELETE operation.

2. ***Cascade:*** When deleting a Tuple, delete all Tuples that reference the deleted Tuple through a Foreign Key.

3. ***Set Default:*** Also known as Set Null. When deleting a Tuple, the Foreign Key values of all other tuples which reference it are set to NULL or a default if possible.

With UPDATE operation, if the Foreign Key value is invalid, the operation is rejected or the Foreign Key value is set to NULL or a default if possible. When changing the Primary Key value of a Tuple referenced by other Tuples with UPDATE, the ***Restrict***, ***Cascade*** and ***Set Default*** options are also available.

<u>Check Constraints and Business Rules</u>
Check Constraints and Business Rules ensure that data fits the user's expectations of how the business should run. These constraints are written by the database designer, and are enforced by code in Applications implementing the Database. These constraints *cannot* be directly expressed in the Schemas of the data model.

2.3 CONVERTING E-R/CONCEPTUAL DATABASE INTO A RELATIONAL/LOGICAL DATABASE

This section will be an implementation of converting a Conceptual (E-R Model) Database to a Logical (Relational Model) Database for our *California Aid Database*. All Entity and Relationship Types will be converted into Relational Schema. Constraints will be created to preserve the validity of the Relational Database while fitting the needs of our Organization. Sample Tuples will be created for each Relation to illustrate how the Relational Database Model will function in the real world.

2.3.1 RELATION SCHEMA FOR LOCAL DATABASE

The following section will be a listing of each Relation Schema for our Relational Model. Attributes, Entities and Relationships from the E-R Model will be represented, as well as the appropriate constraints and Keys.

Relation Schema: employee
employee(**EmployeeID**, SSN, Hire Date, End Date fName, mName, lName, Street, City, State, Zip, Phone Number, Sex, DepatmentID)

Attributes:

EmployeeID	Integer, 0 to Max, Primary Key
SSN	Integer, 000000000 - 999999999
Hire Date	Date
End Date	Date
fName	varchar2(255)
mName	varchar2(255)
lName	varchar2(255)
Street	varchar2(255)
City	varchar2(255)
State	varchar2(2)
Zip	Integer, 00000-99999
Phone Number	Integer, 0000000000-9999999999
Sex	varchar2(1) ‘M’ or ‘F’
DepartmentID	Integer; 2,4,6

Candidate Keys: EmployeeID (primary Key), SSN
Primary Keys/Entity Integrity Constraint:
EmployeeID must be unique and cannot be NULL.
Uniqueness Constraint: SSN must be unique
Referential Integrity Constraint: DepartmentID is a Foreign Key for Employee.
Business Constraint: Employee can only work for one Department.
Derivation From Entity and Relationship Types:
Derived from the Employee Entity Type. DepartmentID is the attribute from Department used as a Foreign Key for the *Works For* Relationship. Composite attributes have been broken into simple components.

Relation Schema: Student
student(**StudentID**, SSN, Academic standing, fName, mName, lName, Street, City, State, Zip, Phone Number, Sex, Income Status)

Attributes:

StudentID	Integer, 0 to Max, Primary Key
SSN	Integer, 000000000 - 999999999
Academic Standing	varchar2(1), 'G' or 'B'
fName	varchar2(255)
mName	varchar2(255)
lName	varchar2(255)
Street	varchar2(255)
City	varchar2(255)
State	varchar2(2)
Zip	Integer, 00000-99999
Phone Number	Integer; 00000000000-99999999999
Sex	varchar2(1), 'M' or 'F'
Income Status	varchar2(255), "Dependent" or "Independent"

Candidate Keys: StudentID(Primary Key), SSN
Primary Key/Entity Integrity Constraint: StudentID must be unique and cannot be NULL
Referential Integrity Constraint: None
Uniqueness Constraint: SSN must be unique
Business Constraint: All students must be attending a School or have attended a School in the history of the Organization's records.
Derivation From Entity and Relationship Types: Derived from the Student Entity Type. Composite attributes have been broken into simple components.

Relation Schema: Parent
parent(**SSN**, fName, mName, lName, Street, City, State, Zip, Phone Number, Birthday, Sex, Status, Income)

Attributes:

SSN	Integer, 000000000 - 999999999
fName	varchar2(255)
mName	varchar2(255)
lName	varchar2(255)
Street	varchar2(255)
City	varchar2(255)
State	varchar2(2)
Zip	Integer, 00000-99999
Phone Number	Integer, 00000000000-99999999999
Birthday	Date
Sex	varchar2(1), 'M' or 'F'
Status	varchar2(255), 'Taxpayer' or 'Non-Taxpayer'
Income	Double, 0 - 999,999.00

Candidate Keys: SSN(primary Key)
Primary Key/Entity Integrity Constraint: SSN must be unique and cannot be NULL
Referential Integrity Constraint: None
Business Constraint: None
Derivation From Entity and Relationship Types: Derived from the Parent Entity Type. Composite attributes are broken into their simple components.

Relation Schema: Department
department(<u>**DepartmentID**</u>, Name)

Attributes:

<u>**DepartmentID**</u>	Integer; 2, 4, 6
Name	varchar2(255)

Candidate Keys: DepartmentID(primary Key)
Primary Key/Entity Integrity Constraint: DepartmentID
Referential Integrity Constraint: None
Business Constraint: None
Derivation From Entity and Relationship Types: Derived from the Department Entity Types. DepartmentID distinguishes departments from one another. This is a Superclass for the Logistics, Outreach, and Review Board Subclasses. The Primary Key will be used as Foreign Key for the Subclasses.

Relation Schema: Logistics
logistics(**DepartmentID**)

Attributes:

<u>**DepartmentID**</u>	Integer Constant, 2

Candidate Keys: DepartmentID(Primary Key)
Primary Key/Entity Integrity Constraint: DepartmentID must have a value of 2, cannot be changed and cannot be NULL.
Referential Integrity Constraint: DepartmentID must exist in a Department Tuple.
Business Constraint: Logistics can only have one value for the DepartmentID
Derivation From Entity and Relationship Types: Derived from the Logistics Entity Type. Subclass of Department, implementing the Multiple Relations - Superclass and Subclass approach for Mapping Superclasses and Subclasses. This required us to drop the AccessCode attribute from our E-R Model and use the DepartmentID instead.

Relation Schema: Outreach
Outreach(**DepartmentID**)
Attributes:

DepartmentID	Integer Constant, 4

Candidate Keys: DepartmentID(Primary Key)
Primary Key/Entity Integrity Constraint: DepartmentID must have a value of 4, cannot be changed and cannot be NULL.
Referential Integrity Constraint: DepartmentID must exist in a Department Tuple.
Business Constraint: Outreach can only have one value for the DepartmentID
Derivation From Entity and Relationship Types: Derived from the Outreach Entity Type. Subclass of Department, implementing the Multiple Relations - Superclass and Subclass approach for Mapping Superclasses and Subclasses. This required us to drop the AccessCode attribute from our E-R Model and use the DepartmentID instead.

Relation Schema: Review Board
review board(**DepartmentID,** BudgetID*)

Attributes:

DepartmentID	Integer Constant, 6
BudgetID*	Integer; 0 to Max

Candidate Keys: DepartmentID(Primary Key)
Primary Key/Entity Integrity Constraint: DepartmentID must have a value of 6, cannot be changed and cannot be NULL.
Referential Integrity Constraint: DepartmentID must exist in a Department Tuple.
Business Constraint: Review Board can only have one value for the DepartmentID
Derivation From Entity and Relationship Types: Derived from the Review Board Entity Type. Subclass of Department, implementing the Multiple Relations - Superclass and Subclass approach for Mapping Superclasses and Subclasses. This required us to drop the AccessCode attribute from our E-R Model and use the DepartmentID instead.
BudgetID is a Foreign Key of Budget Relation for the Relationship ‘Checks’.

Relation Schema: School
school(**SchoolID,** Name, Street, City, State, Zip, DepartmentID*)

Attributes:

SchoolID	Integer, 0 to MaxID
Name	varchar2(255)
Street	varchar2(255)
City	varchar2(255)
State	varchar2(255)
Zip	Integer, 00000-99999
DepartmentID	Integer constant, 4;

Candidate Keys: SchoolID (Primary Key)
Primary Key/Entity Integrity Constraint: SchoolID must be unique and cannot be NULL.
Referential Integrity Constraint: DepartmentID is a Foreign Key for Department.
Business Constraint: None
Derivation From Entity and Relationship Types: Derived from School Entity Type. DepartmendID is a Foreign Key of Outreach Relation for the 'Communicates with' Relationship.

Relation Schema: Application
application(**AppID,** requestedAmount, Date, Status, approved)

Attributes:

AppID	Integer, 0 to MaxID
requestedAmount	Double, (0 - 99,999.00)
Date	Date
Status	varchar2(255), 'Complete' or 'Incomplete'
approved	Boolean, True or False

Candidate Keys: AppID(Primary Key)
Primary Key/Entity Integrity Constraint: AppID must be unique and cannot be NULL.
Referential Integrity Constraint: None
Business Constraint: Status can only be either 'Complete' or 'Incomplete'
Derivation From Entity and Relationship Types: Dervied from the Application Entity Type.

Relation Schema: Data
data(**<u>DataID</u>, <u>Description</u>**, Date, DepartmentID*)

Attributes:

<u>DataID</u>	Integer, 0 - MAX
<u>Description</u>	varchar2(255), 'Parent' or 'Student'
Date	Date
DepartmentID*	Integer Constant; 2

Candidate Keys: DataID(Primary Key)
Primary Key/Entity Integrity Constraint: DataID must be unique and cannot be NULL.
Referential Integrity Constraint: DepartmentID is a Foreign Key from Logistics Relation for the 'are collected by' Relationship.
Business Constraint: Description attribute can only be 'Parent' or 'Student'
Derivation From Entity and Relationship Types: Derived from the Data Entity Type. DepartmentID is a Foreign Key from Logistics for the 'are collected by' relationship. This means that the DepartmentID must be a value of 2.

Relation Schema: Information
information(**<u>SourceID</u>,** Date, DepartmentID*)

Attributes:

<u>SourceID</u>	Integer, 0 to MaxID
Date	Date
DepartmentID*	Integer Constant; 6

Candidate Keys: SourceID(Primary Key)
Primary Key/Entity Integrity Constraint: SourceID must be unique and cannot be NULL.
Referential Integrity Constraint: DepartmentID is a Foreign Key from Review Board for the 'is reviewed by' Relationship. The value must be 6.
Business Constraint: None
Derivation From Entity and Relationship Types: Derived from the Information Entity Type. DepartmentID is a Foreign Key from Review Board for the 'is reviewed by' relationship. This means that the DepartmentID must be a value of 6.

Relation Schema: Budget
budget(**<u>BudgetID</u>,** Amount, Date)

Attributes:

<u>BudgetID</u>	Integer, 0 to MaxID
Amount	Double, (0.00 to 999, 999,999.00)
Date	Date

Candidate Keys: BudgetID (Primary Key)
Primary Key/Entity Integrity Constraint: BudgetID must be unique and cannot be NULL.
Referential Integrity Constraint: None
Business Constraint: None
Derivation From Entity and Relationship Types: Derived from the Budget Entity Type.

Relation Schema: Financial Aid Package
financial aid package(**<u>packageID</u>,** Type, Amount, BudgetID*)

Attributes:

<u>packageID</u>	Integer, 0 to MaxID
Type	varchar2(255); 'CalGrantA', 'CalGrantB', 'Other'
Amount	Double, (0.00 to 999, 999,999.00)
BudgetID*	Integer; 0 to Max;

Candidate Keys: packageID(Primary Key)
Primary Key/Entity Integrity Constraint: packageID must be unique and cannot be NULL.
Referential Integrity Constraint: BudgetID is a Foreign Key from Budget Relation for the 'allocates funds to' relationship.
Business Constraint: Package *Type* attribute can only be 'CalGrantA', 'CalGrantB', or 'Other'.
Derivation Frotity and Relationship Types: Derived from the Financial Aid Package Entity Type. Contains the Primary Key from Budget Relation as a Foreign Key for the 'allocates funds to' Relationship.

Relation Schema: Uses info from
uses info from (**StudentID**, **ParentSsn**)

Attributes:

StudentID	integer, 0 to Max
ParentSsn	integer, 0 to Max

Candidate Keys: [StudentID, ParentSsn]
Primary Key/Entity Integrity Constraint: The combination of StudentID and ParentSsn is unique and cannot be NULL.
Referential Integrity Constraint: StudentID is a Foreign Key from Student, ParentSsn is a Foreign Key from Parent (renamed Ssn to ParentSsn).
Business Constraint: None
Derivation From Entity and Relationship Types: Derived from the N:1 Student *uses info from* Parent using the ***cross-reference approach***. *uses info from* is a Relationship Relation with Foreign Keys for Student and Parent.

Relation Schema: Fills Out
Fills Out (**StudentID, ApplicationID**, Date)

Attributes:

StudentID	Integer; 0 to Max
ApplicationID	Integer; 0 to Max
Date	Timestamp

Candidate Keys: [StudentID, ApplicationID]
Primary Key/Entity Integrity Constraint: The combination of StudentID and ApplicationID is unique and cannot be NULL.
Referential Integrity Constraint: StudentID is a Foreign Key from Student, ApplicationID is a Foreign Key from Application.
Business Constraint: Fills out needs a Date to indicate the time and day the Application was filled out.
Derivation From Entity and Relationship Types: Derived from the N:1 Student *Fills Out* Application using the ***cross-reference approach***. *Fills Out* is a Relationship Relation with Foreign Keys from Student and Application.

Relation Schema: Becomes
becomes(**AppID**, **DataID**, **DataDescription**,Date)
Attributes:

AppID	Integer; 0 to Max
DataID	Integer; 0 to Max
DataDescription	varchar2(255); 'Parent' or 'Student'
Date	timestamp

Candidate Keys: [AppID, DataID, DataDescription]
Primary Key/Entity Integrity Constraint: The combination of AppID, DataID, and DataDescription is unique and cannot be NULL.
Referential Integrity Constraint: DataID and DataDescription are both Foreign Keys from Data Relation. DataDescription has been renamed from 'Description' of Data Relation.
Business Constraint: A timestamp is required to notify when an Application became Data.
Derivation From Entity and Relationship Types: Derived from the M:N Application *Becomes* Data using the ***cross-reference approach***. *Becomes* is a Relationship Relation with Foreign Keys for Application and Data.

Relation Schema: Uses and Stores Into
uses and stores into(**DataID**,**DataDescription**, **SourceID**, SourceDate)
Attributes:

DataID	Integer; 0 to Max
DataDescription	varchar2(255); 'Parent' or 'Student'
SourceID	Integer; 0 to Max
SourceDate	timestamp

Candidate Keys: [DataID, DataDescription, SourceID]
Primary Key/Entity Integrity Constraint: The combination of DataID, DataDescription, SourceID are unique and cannot be NULL.
Referential Integrity Constraint: DataID, DataDescription, SourceID are all Foreign Keys from Data and Information.
Business Constraint: A sourceDate is needed to indicate when Data was used and stored into Information.
Derivation From Entity and Relationship Types: Derived from the 1:N:M *Uses and Stores Into* Relationship using the ***cross-reference approach***. *Uses and Stores Into* is a Relationship Relation. Although Logistics is the Department that does this Activity, the Key of Logistics is not Required due to Total Participation.

Relation Schema: Distributed To
distributed to (**StudentID**, **faPackageId**, Date)
Attributes:

StudentID	Integer; 0 to Max
faPackageID	Integer; 0 to Max
Date	timestamp

Candidate Keys: [StudentID, faPackageID]
Primary Key/Entity Integrity Constraint: The combination of StudentID and faPackageID are unique and cannot be NULL.
Referential Integrity Constraint: StudentID is a Foreign Key from Student and faPackageID is a Foreign Key from Financial Aid Package.
Business Constraint: A timestamp is required to know when a Financial Aid Package was Distributed to a Student.
Derivation From Entity and Relationship Types: Derivation from the M:N Financial Aid Package *Distributed To* Student using the ***cross-reference approach***. *Distributed To* is a Relationship Relation with Foreign Keys for Financial Aid Package and Student.

Relation Schema: Attending
Attending (**StudentID**, **SchoolID**)
Attributes:

StudentID	Integer; 0 to Max
SchoolID	Integer; 0 to Max

Candidate Keys: [StudentID, SchoolID]
Primary Key/Entity Integrity Constraint: The combination of StudentID and SchoolID are unique and cannot be NULL.
Referential Integrity Constraint: StudentID is a Foreign Key from Student and SchoolID is a Foreign Key from School.
Business Constraint: None
Derivation From Entity and Relationship Types: Derived from the N:M Student *Attending* School using the ***cross-reference approach.*** *Attending* is a Relationship Relation with Foreign Keys for Student and School.

2.3.2 SAMPLE DATA OF RELATION

Now that we have described each Relation Schema for our Relational Model, we will introduce a list of Tuples that belong to hypothetical Relation States for each Relation Schema in our Organization's Database. The tuples will be listed in a table format, Relational Schema Attributes are columns, individual Tuples are rows. Each Relation corresponding to Entity Sets will be around 10 Tuples. Relations which correspond to Relationship Sets will have between 60 and 100 Tuples.

Employee

EmployeeID	SSN	Hire Date	End Date	fName	mName	lName
7297472310	892-29-8273	9/19/2009	8/21/2016	Ruby	Marie	Austin
8634855525	366-91-4687	2/7/2011	4/18/2016	Evelyn	Doris	Romero
7653035962	202-17-9739	4/14/2011	12/13/2015	Rebecca	Mildred	Crawford
7432861030	673-54-2516	11/22/1992	9/15/2016	Laura	Cynthia	Perkins
5136103217	265-35-3631	7/6/2002	12/12/2015	Irene	Cynthia	Gordon
1790837562	369-39-8491	6/7/2010	5/4/2016	Diane	Ann	Palmer
9050573884	627-43-2504	8/27/2007	3/21/2016	Christopher	Brandon	Hunt
6072620691	289-41-3669	4/21/2010	1/23/2016	Joyce	Rachel	Morales
1254527508	681-30-2291	3/2/2002	7/22/2016	Susan	Nancy	Barnes
9056715429	655-71-1337	7/12/2009	5/18/2016	Gregory	Jonathan	Hayes

Street	City	State	Zip	phone Number	Sex	Department-ID
12 Mccormick	Marietta	California	59605	1-(770)729-4867	F	1
5628 Luster Lane	Shawnee Mission	California	19083	1-(913)568-7709	F	1
65 Orin Park	Phoenix	California	36009	1-(602)963-5974	F	2
8 Jenifer Center	Alexandria	California	81069	1-(571)616-4125	F	1
8 Walton Place	Schenectady	California	77359	1-(518)371-8288	F	3
5628 Luster Lane	Aiken	California	17381	1-(803)513-1468	F	2
65 Orin Park	Chula Vista	California	91464	1-(619)274-1281	M	3
8 Jenifer Center	Seattle	California	89597	1-(360)779-4964	F	2
8 Walton Place	Prescott	California	58025	1-(520)483-4908	F	3
78 Autumn Leaf	Richmond	California	36790	1-(804)751-1750	M	1

Student

StudentID	SSN	Academic Standing	fName	mName	lName
1790837562	759-71-3071	G	Julia	Janet	Rivera
7297472310	121-25-6054	B	Evelyn	Sara	Lawrence
9050573884	605-05-7166	B	Diana	Irene	Morgan
8634855525	681-26-4075	B	Fred	Shawn	Wells
6072620691	388-64-7682	B	Russell	Willie	Davis
7653035962	154-21-2563	B	Andrea	Lisa	Jordan
1254527508	972-50-6097	B	Justin	Gregory	Dixon
7432861030	903-22-4530	G	Joseph	Gary	Allen
9056715429	943-36-7126	B	Amy	Andrea	Harvey
5136103217	499-33-8215	B	Jesse	Bobby	Carr

Street	City	State	Zip	Phone Number	Sex	Income Status
75329 Kingsford Place	Lincoln	California	93664	1-(402)722-8928	F	Dependent
292 Gina Parkway	Peoria	California	90567	1-(309)564-7069	F	Dependent
73735 Old Gate Plaza	Las Vegas	California	83668	1-(702)258-0913	F	Independent
5628 Luster Lane	Rockford	California	57810	1-(815)828-3018	M	Independent
85427 Hanover Hill	Asheville	California	46979	1-(828)799-6661	M	Dependent
65 Orin Park	Silver Spring	California	92365	1-(703)801-2819	F	Independent
40 American Alley	Jackson	California	59628	1-(601)941-6830	M	Dependent
8 Jenifer Center	Modesto	California	75525	1-(209)154-0895	M	Dependent
47659 Vernon Parkway	San Antonio	California	49940	1-(210)155-8356	F	Dependent
8 Walton Place	Alexandria	California	40925	1-(318)137-6571	M	Dependent

Parent

SSN	fName	mName	lName	Street	City
891-40-6987	Christopher	Shawn	Hanson	6 Trailsway Pass	West Hartford
833-93-4621	Frank	Christopher	Bowman	22324 Welch Avenue	Houston
923-56-2201	Jesse	Michael	Gutierrez	28557 Esch Street	Charleston
189-82-9167	Margaret	Melissa	Vasquez	75 Magdeline Circle	Cleveland
562-66-5340	Jean	Rachel	Mccoy	588 Columbus Place	Dallas
481-79-1400	Carl	Aaron	Harrison	2 Aberg Alley	Ogden
284-64-0104	Alan	Martin	Romero	38 New Castle Street	Hamilton
603-26-4533	Phillip	Jose	Lee	2 Pennsylvania Park	Jacksonville
891-16-0797	Ernest	Howard	Perkins	727 Arapahoe Terrace	Tacoma
108-65-5087	Louise	Margaret	Howell	7 Northview Way	Houston

State	Zip	Phone Number	Birthday	Sex	Status	Income
California	76070	1-(860)409-0876	7/29/1947	M	Taxpayer	$182,250.41
California	60242	1-(281)597-0976	8/31/1943	M	Taxpayer	$92,403.07
California	78109	1-(304)118-2496	1/16/1963	M	Non-Taxpayer	$29,997.39
California	65877	1-(216)669-3115	10/18/1955	F	Taxpayer	$169,984.39
California	95775	1-(214)775-1208	11/14/1983	F	Non-Taxpayer	$203,505.90
California	97041	1-(801)466-8577	3/9/1973	M	Taxpayer	$3,510.55
California	97326	1-(937)275-4442	2/14/1972	M	Non-Taxpayer	$69,562.15
California	91971	1-(904)455-1993	1/21/1990	M	Non-Taxpayer	$43,023.08
California	31379	1-(253)902-0077	6/9/1949	M	Non-Taxpayer	$1,276.28
California	15416	1-(713)354-0788	8/29/1958	F	Non-Taxpayer	$106,920.72

Department

DepartmentID	Name
2	Logistics
4	Outreach
6	Review Board
2	Logistics
2	Logistics
4	Outreach
6	Review Board
2	Logistics
4	Outreach
6	Review Board

Logistics

DepartmentID
2
2
2
2
2
2
2
2
2
2

Outreach

DepartmentID
4
4
4
4
4
4
4
4
4
4

Review Board

DepartmentID	BudgetiD
6	4272770472
6	9059287380
6	9298575036
6	9187532424
6	7939665891
6	8210779778
6	6908035933
6	4732476272
6	3326301453
6	5785782232

School

SchoolID	Name	Street	City	State	Zip	DepartmentID
9048669109	Kim High School	257 Fuller Plaza	Trashigang	California	71189	4
8717238415	Swallow High School	3255 Ludin Tr	Emporeío	California	45151	4
1160732741	Meadow High School	967 Talis Street	Lajinha	California	50252	4
9588202017	Del Sol High School	255 Dex Circle	Jedlnia-Letnisko	California	69131	4
9703336869	Dixon High School	1698 Artisan Rd	Pasir Mas	California	88412	4
5085405211	Luster High School	8 Graedel Point	Oemofa	California	93464	4
2449147802	Men High School	47 Crest Lane	Surin	California	91640	4
1942017597	Victoria High School	5 2nd Crossing	Wanzu	California	35953	4
3994605108	Hansons High School	9 Cordelia Way	Semambung	California	19031	4
9840715155	Lukken High School	95 Emmet Drive	Guintubhan	California	76061	4

Application

AppID	requestedAmount	Date	Status	Approved
5573810649	$151,751.21	10/6/2004	Complete	TRUE
2718413728	$112,822.16	9/26/2005	Complete	TRUE
3237891270	$37,884.78	12/28/2006	Incomplete	FALSE
8179517399	$60,172.34	9/26/2009	Incomplete	FALSE
1871486727	$50,320.60	4/24/2003	Incomplete	FALSE
8933199605	$104,723.30	1/17/2015	Incomplete	FALSE
3367648435	$127,559.54	8/18/2009	Complete	TRUE
7303644633	$90,446.11	5/3/2010	Incomplete	FALSE
3397254472	$73,345.11	11/4/2002	Complete	TRUE
4957166271	$138,262.88	10/13/2003	Complete	FALSE

Data

DataID	Description	Date	DepartmentID
8173167540	Parent	1/31/2005	2
1961415273	Parent	2/2/2012	2
4742667123	Student	1/21/2002	2
3016457484	Parent	3/27/2014	2
2557609258	Student	12/28/2013	2
2108832567	Parent	6/26/2016	2
9650932693	Student	10/23/2013	2
9114524874	Student	1/13/2010	2
3619542131	Parent	2/10/2011	2
6214618179	Student	1/7/2011	2

Information

SourceiD	Date	DepartmentID
2140679661	11/12/2000	6
9356043878	11/3/2001	6
4846479615	4/15/2014	6
5321963646	12/13/2003	6
9447865377	5/12/2014	6
7017276710	10/27/2009	6
7709930036	3/9/2006	6
5074368872	2/4/2011	6
5345338463	1/24/2015	6
8943067031	1/26/2008	6

Budget

BudgetiD	Amount	Date
2752766764	$149,388,643.60	6/2/2007
8801886598	$694,383,234.80	5/29/2006
8140074021	$808,261,289.82	11/10/2009
1399589692	$824,257,957.30	10/21/2004
7331632521	$751,025,727.68	7/27/2009
5245460183	$452,400,075.31	3/25/2008
6761872466	$562,657,725.06	11/15/2003
5823153509	$275,270,391.60	7/30/2009
9910849373	$861,546,782.24	6/25/2016
2017309389	$608,778,874.57	6/18/2016

Financial Aid Package

packageID	Type	Amount	BudgetID
5254601745	CalGrantA	$80,105.40	4255623183
6105570425	CalGrantB	$86,601.01	3993562388
6348919004	Other	$133,091.96	9438483807
7762351110	CalGrantA	$96,025.01	3439169834
1419813281	CalGrantB	$139,719.31	3505003159
1419816081	Other	$63,640.37	6813535213
2254137750	CalGrantA	$13,285.95	8035973758
5140294121	CalGrantB	$103,429.47	6546288496
3577914829	Other	$75,404.32	6115168822
5063199001	CalGrantA	$75,576.05	2223663280

Uses Info From

Uses info from	
StudentID	**ParentSsn**
6244661811	929-94-0413
4291103390	610-94-8351
8626814464	507-67-6953
8872359664	265-25-8271
9054634170	898-63-7481
4845888468	181-74-4073
6110924932	724-73-5852
9421987319	258-61-5297
5491069483	648-93-2126
2155008039	695-25-5795
5983496795	655-99-3261
9843249348	701-61-7067
9560509717	820-77-2468
4356292990	729-55-3760
8570130163	424-03-4537
8877054113	257-85-5855
4608773902	663-60-4980
7157022598	762-34-3847
8256034195	377-12-4429
3872331697	523-82-4678
8625466153	882-94-7278
1240464940	207-53-2249
7809470539	210-87-3232
7054419882	681-50-9839
2861698683	532-00-1970
2371386066	934-06-0067
3141274967	213-33-0258
2212261869	756-81-8143
1382429173	866-58-8885
1905321679	986-44-0754
4806762250	394-24-4305
7059426134	283-37-1508
7251171537	786-64-8307
4145271695	557-70-2489
9525269965	571-40-9138
2060727734	991-29-3464
3861155194	501-37-5745
7865395804	284-55-2214
3345925346	984-12-7899
5006504100	911-41-7313
8945638618	690-02-7470
1180443628	415-64-7947
3360781969	723-98-0725
8116105810	922-42-1954
3641188562	144-95-4067
5222327449	387-50-5141
2912988288	316-93-0130
5146818751	647-83-2928
2134195344	649-74-1190
9143093599	559-07-5808
1426766808	939-04-8455
8370356087	689-39-4749
4163686476	864-96-7965
4158040021	304-33-5616
2289435313	578-10-4939
4008782515	239-98-2450
9597975123	999-44-1332
2787203142	637-33-5529
2826512506	475-30-0501
7989577964	715-88-1267
3175366923	637-61-0023
7983083127	573-16-8576
1947530939	899-75-6947
4576823779	392-18-3526
9683734061	432-94-6010
6736720065	474-99-6712
6230549539	130-41-4172
8280095779	388-23-1527
9814890054	970-31-2137
2020754207	714-26-3856
7669261555	119-25-3357
7255913435	429-89-6305
5354106794	459-13-4579
7774777808	265-89-5829
7170894968	747-41-8567
1355463354	355-78-1543
5897667602	230-13-1887
8877137023	972-31-6234
6666048627	783-97-3035
7981185818	337-45-4429
2526525552	518-29-7123
9526604614	156-37-9515
5943874223	633-39-2246
8995750830	730-99-4815
3142665503	324-77-4183
5452785914	231-47-0236
8686183828	205-15-1126
7408823957	648-29-3215
8563133447	819-49-2879
3409579993	997-84-8135
7203110209	817-24-2501
4795619377	435-02-3410
3870112516	536-60-8174
8008103359	628-36-7972
8602495047	730-75-4880
1223986377	689-01-7367
7833993404	398-88-6228
1111310805	741-35-0395
5502692287	906-20-7882
8389795848	553-02-7285

<u>Fills out</u>

fills out

StudentID	ApplicationID	Date
2695973923	545109594	11/18/2010
8927607779	760986089	6/8/2013
8671040042	500135733	2/28/2000
4196526037	744885746	2/26/2000
2418774386	875006184	5/12/2002
4800282783	634750653	4/6/2014
6461008318	850843840	7/2/2004
4859111193	452944320	10/11/2010
1363711087	667301005	3/24/2016
5668104604	586865900	8/19/2004
9761988361	384669154	10/5/2012
5835818087	706710530	8/28/2009
7957876287	370666057	1/17/2008
5377211939	127281684	5/7/2005
6978790017	806939743	10/12/2007
7489536670	814836292	3/30/2011
5487656740	291880178	5/6/2012
2420537813	677532885	12/15/2006
8089189346	731991931	4/6/2007
3764441944	409936597	7/9/2006
7008733739	455147066	3/17/2001
9986183792	371892555	5/7/2005
7027004967	872883266	7/29/2005
6864976411	153393897	11/15/2001
1149294143	256744338	7/11/2014
9682645056	203482989	1/14/2013
3145676410	895388618	7/2/2014
4383560135	327533393	8/17/2008
1755405783	535841523	10/7/2007
1448160680	587009584	8/23/2014
8908712951	551360485	10/2/2000
8381072564	948301523	2/2/2014
5097523393	247801812	2/13/2012
8349873615	237066928	10/2/2001
1886636607	698731046	6/4/2007
6789511393	174927114	2/25/2006
6350742608	902928327	5/30/2002
6793509217	481547012	10/7/2003
4513718691	212187475	10/27/2004
5376555674	121799312	12/8/2015
4782549764	916617050	11/10/2006
8642504438	261401072	5/27/2002
2365093643	531454603	4/25/2009
4638860666	663378049	12/30/2001
3577437430	644715151	1/27/2000
7294748922	482931062	8/10/2005
8162332801	563535853	9/6/2010
2163438204	865087919	5/14/2011
6608799464	172366036	1/10/2003
9793065459	203544986	7/2/2001
9096030482	562853115	7/4/2004
5439430935	377894377	9/11/2008
4093618134	801853789	10/3/2015
8110724054	870751673	1/15/2016
9492063015	463216848	10/2/2008
5560547196	158472859	10/31/2014
8967276869	226633236	1/23/2001
6543400125	435153110	8/7/2007
8062362403	552516718	10/5/2005
4612109961	474937614	9/13/2015
6400397352	749299223	8/5/2002
4546724171	323035140	11/6/2014
9241373081	531052046	1/21/2008
3493108472	584108513	10/17/2001
1924530451	613366782	11/29/2015
3344462051	909251597	2/17/2002
2614739999	389990825	11/7/2000
1132684630	222762717	2/11/2004
4598919704	722826340	9/22/2015
1500617998	432167154	7/3/2006
8906875678	742440607	6/22/2004
4446368372	479199386	8/12/2012
4027958042	452391976	10/4/2002
2225134767	721991553	7/5/2008
2923842147	315304816	4/6/2010
3167942077	611875731	5/25/2008
7170397773	286636799	4/12/2005
5947887808	387896323	1/7/2006
6409773634	656564093	10/9/2012
7124026440	581847372	11/8/2007
6806347325	490749022	12/24/2001
5531233558	717046607	2/6/2003
6863734560	243076388	11/17/2003
8899796988	406697060	12/8/2015
5289048290	220718879	12/20/2003
6982173487	611210352	6/24/2000
9590480114	294403484	3/5/2003
1326240745	595315958	9/29/2005
5795084477	471599047	10/15/2011
2059883669	383462060	7/3/2001
8179680137	912078530	11/28/2009
4709769549	852889715	3/2/2002
5256827959	112365469	1/17/2012
6107465664	443897287	11/17/2010
3542220787	976649185	1/10/2014
3618814078	561115343	4/27/2014
5897909744	455630203	9/23/2000
4756157652	904733588	4/18/2005
7631198779	541799690	11/10/2003
1229354018	500184358	4/6/2007

Becomes

becomes			
AppID	DataID	ataDescrip	Date
8470422757	192911224	Parent	5/8/2005
6306243361	145791980	Student	12/25/2007
9929112813	954955906	Student	9/28/2002
1672494314	160865147	Parent	6/7/2004
9132221646	184039734	Student	12/7/2014
7758265460	994194084	Student	5/31/2015
9958689507	140415034	Student	6/28/2011
9351393091	270665907	Parent	5/11/2007
1531646735	273310470	Student	5/12/2012
9621082833	218981010	Parent	9/2/2003
8565119679	730371967	Student	6/6/2011
4259001679	947837543	Student	7/11/2011
6774545266	941858081	Parent	2/18/2007
4771345544	619933573	Parent	8/24/2006
1263632486	618480894	Student	5/8/2006
5512919645	837283173	Parent	6/21/2012
2191452381	118256210	Student	10/27/2003
2940612042	830412856	Student	1/15/2007
2947720702	838206088	Student	8/19/2010
8673671972	821854645	Student	4/25/2004
8150359881	385103144	Student	4/13/2013
4497808867	387945584	Student	11/15/2011
4080032471	603443655	Student	6/29/2002
3438900904	446059926	Student	11/2/2014
2693667473	472153139	Student	8/14/2012
5373816782	441792296	Parent	10/5/2012
4660999943	622144067	Student	4/7/2008
2820402758	610678139	Parent	1/3/2011
5883896107	673605952	Parent	9/1/2015
7264008984	151371003	Parent	9/24/2002
6487383817	973622894	Parent	4/13/2002
2446776646	549530741	Student	12/12/2013
5843498990	652438048	Student	7/3/2007
9443569440	620634330	Student	6/14/2011
9177987314	315554047	Parent	3/22/2012
8235979745	547196078	Student	9/24/2014
5237654170	461985519	Student	6/12/2006
2694933914	629651440	Parent	9/1/2016
2610418704	235617667	Student	6/28/2013
2062838653	755599559	Student	7/31/2009
5262669066	828749177	Student	1/6/2016
8347908243	849197864	Parent	10/25/2002
6058993919	310502734	Student	3/9/2002
3563587075	488329826	Student	12/1/2008
8431824030	762068706	Parent	2/16/2016
8780461383	589998932	Student	5/23/2013
6355823764	349055095	Student	7/21/2005
9758091775	513380389	Student	6/30/2003
7023824075	230731818	Student	2/11/2010
1607616725	867741041	Student	6/12/2007
9082775205	378795995	Student	12/25/2007
3605766240	480518614	Student	10/11/2016
3649517433	866361307	Student	4/10/2007
1867780200	823327334	Student	10/27/2000
7387358868	111493380	Student	1/1/2012
9375701420	468432112	Parent	9/27/2016
4188883511	923296030	Student	9/4/2014
2050760040	267770855	Student	6/12/2013
8737552625	780365665	Student	1/24/2011
5138380300	736863347	Student	4/17/2002
1870749910	629738321	Student	2/18/2001
8481843159	708616149	Student	8/11/2006
2848129798	171937398	Student	2/13/2005
3234098625	662449464	Student	10/8/2016
1573845496	790801289	Parent	6/26/2011
4545623148	830451695	Student	12/18/2014
7464201801	134012138	Student	6/27/2002
2940759469	268610252	Parent	3/29/2009
9436931259	388386374	Student	7/16/2016
7741138026	476749021	Student	5/21/2013
3275658358	837351901	Student	2/16/2009
7704786536	380462495	Parent	6/21/2004
7758007281	490624243	Student	7/9/2013
9278766381	932639380	Student	4/7/2002
8091994934	791425201	Student	7/8/2013
7049051944	867928646	Student	7/15/2011
2630652312	460620021	Parent	9/2/2000
9410251634	620849468	Student	1/4/2008
9599931710	245083764	Student	11/26/2010
6123661046	262405070	Parent	3/11/2010
6890472417	548941686	Student	4/10/2001
3293368086	262190411	Student	11/1/2008
1726150962	988936418	Student	3/1/2003
5157745554	626100739	Parent	12/7/2004
6797623965	511381727	Student	9/12/2016
9956175429	421328708	Parent	7/18/2010
5097145615	668523996	Student	3/5/2008
5169340344	135537766	Student	6/12/2000
7020621560	512288959	Parent	2/9/2014
9533499489	720397181	Parent	11/20/2007
4793542738	791715209	Student	1/21/2003
2368809859	658294990	Student	9/8/2001
1846552269	327064016	Student	6/3/2005
8113418835	693606695	Parent	12/26/2005
4873638747	621081291	Student	2/3/2016
5408502056	343234820	Parent	3/2/2011
9923356434	211531151	Student	8/17/2001
8334037926	305335467	Student	10/5/2016
2021426938	639634110	Student	3/30/2001
6501721898	248736565	Parent	4/8/2012

Uses and Stores Into

Uses and Stores Into			
DataID	DataDescriptio	SourceID	SourceDate
2383460093	Parent	791230759	5/28/2016
4756612540	Student	610886834	3/17/2009
5744067749	Student	975895883	6/7/2001
3724306136	Student	551784756	3/12/2004
9389872187	Parent	410079561	11/10/2002
2914151307	Parent	490659459	11/28/2004
8117051185	Student	890754611	6/3/2010
9498835443	Student	939552556	6/26/2008
2380812792	Student	178901946	6/14/2006
6025184327	Student	926851260	8/18/2011
5961273192	Parent	219278320	2/1/2003
1678366225	Student	747132771	3/7/2008
4247733290	Student	185652941	11/4/2005
4033817583	Student	481288213	3/26/2002
9072291110	Student	948015556	12/5/2003
4282968409	Student	515704334	5/8/2013
6424388069	Parent	486009477	3/10/2004
5432830263	Parent	674916528	9/2/2011
5643179217	Parent	392307262	2/26/2014
5914179540	Student	895847890	5/2/2000
7911490925	Parent	926256280	8/5/2003
7264651124	Student	461462942	7/13/2002
1682503162	Student	308252381	4/5/2015
6869439805	Student	266842544	2/20/2014
8335579686	Parent	722683955	7/15/2016
9035078328	Student	372656119	7/28/2013
7772245718	Student	385212737	6/4/2013
2456059212	Student	880565480	10/13/2016
4584228967	Student	859055331	1/26/2012
6939215264	Student	462258160	11/27/2009
1658430519	Student	434906348	4/4/2013
7389056883	Parent	130132489	6/14/2003
4934387418	Parent	355264683	5/24/2003
2437792037	Student	884541699	11/12/2002
9416170366	Student	417863453	1/3/2015
8669423395	Student	754892195	4/16/2005
2626624706	Student	542959453	5/12/2007
2015845473	Parent	264567552	7/27/2011
2018876452	Student	339909069	11/12/2009
7456827613	Student	688970683	7/7/2001
4985689712	Student	741511952	4/17/2009
9673710085	Student	576427009	11/20/2005
7666776589	Parent	878182551	4/19/2014
1201750503	Student	225112277	12/11/2012
1613260701	Student	916583633	4/3/2013
2783214265	Parent	171675092	9/7/2003
3127912115	Parent	341389603	2/19/2009
6671643534	Student	796034372	6/20/2004
4624284731	Student	219249181	8/31/2014
7789330809	Student	420812300	12/18/2010
1937434622	Student	777750065	9/17/2008
3492134921	Student	160000861	8/10/2008
6321122308	Student	417435821	2/24/2012
4923707537	Parent	761624154	1/24/2006
5923194175	Student	979914288	7/5/2002
8624906897	Parent	528194856	10/5/2003
4685534090	Student	305217517	6/21/2014
2319818751	Student	427575219	7/31/2015
5996630227	Parent	194925277	2/18/2007
1633608765	Student	551695854	11/18/2005
6952804808	Parent	438125248	2/24/2005
4039906045	Student	709427701	1/14/2015
5213468152	Parent	539176745	7/8/2007
5677209764	Parent	780360729	6/8/2015
3368664956	Student	115292172	10/21/2014
9317851776	Student	760596986	5/19/2012
6859205075	Student	856875340	1/14/2004
3769310168	Parent	501609678	4/7/2013
8998724703	Student	169153483	8/2/2001
5207810282	Parent	595453127	1/20/2003
9114976962	Parent	526385589	7/28/2013
4215491676	Student	874791981	6/14/2016
1863128469	Parent	923957011	7/5/2004
3141479151	Student	193433185	12/24/2013
4399788490	Parent	905300119	1/8/2001
4066450035	Student	263655517	2/14/2008
2049642713	Student	965712400	12/22/2004
1790030899	Parent	283116253	6/29/2007
3957702438	Student	383384857	3/25/2002
6580681272	Student	956986770	2/7/2005
6427276056	Student	295769910	12/30/2006
3729327463	Parent	261316889	12/31/2001
8245728165	Student	523712351	2/11/2004
6574272944	Student	760165736	2/6/2006
9792101763	Student	952325359	3/31/2010
3489806367	Parent	962338335	9/10/2003
7151237500	Student	939660240	3/3/2010
6311979281	Student	333693222	4/23/2008
6877707788	Student	848043118	2/1/2012
8946593863	Parent	340758121	3/14/2005
7856282687	Student	884324619	5/6/2004
8520537697	Student	853413350	7/20/2011
9294316824	Student	411256723	3/18/2001
6893108752	Student	830640747	7/16/2004
3482843020	Student	940428193	1/11/2007
3995774141	Parent	116579213	10/5/2016
9515155834	Student	379527064	7/7/2005
9489972505	Student	169496847	2/14/2014
8146428208	Parent	620251676	7/13/2013
9728437257	Student	372728523	3/13/2007

Distributed to

Distributed To		
StudentID	faPackageID	Date
9699296786	915813670	8/11/2012
3184921302	309068775	1/15/2000
3340323138	325345849	7/9/2005
6027817086	819660427	6/26/2001
8580584215	425828658	10/21/2005
9338904051	932919896	7/7/2015
8788152979	776496253	3/22/2013
3977122800	668870134	7/17/2010
1572812511	150043273	11/6/2009
9976464194	368405934	4/23/2010
1809528007	715743571	3/4/2003
6525028340	286841928	4/16/2005
8171942209	798586747	11/8/2013
1612819507	305301428	12/25/2015
7046305331	661902008	9/19/2012
7439194721	721397224	1/19/2009
7523234968	547532099	3/16/2004
7222722105	311671975	3/11/2011
3551158888	560139793	9/18/2011
5792037722	396862257	7/3/2013
1768884912	503469841	6/5/2013
6345628790	795662912	7/23/2016
3045312680	861494141	11/28/2014
3815688826	346053079	5/11/2015
8688809636	902033884	10/30/2015
3231839303	797824941	8/28/2009
9258598025	639731104	11/29/2001
7628691129	467999242	10/21/2006
2399191008	180910689	3/20/2016
3629330378	172894837	10/17/2007
7014277073	879947478	1/22/2002
4887039669	957745780	9/7/2014
7353476385	444272865	2/8/2012
2836349403	898835139	6/5/2005
3291466403	502546500	7/1/2004
2217752795	434461516	3/20/2007
8645410611	380686562	6/8/2016
6547664017	772370416	10/24/2005
9306344470	633406713	11/1/2015
2292190395	561926131	8/9/2012
2608381894	752466334	8/8/2009
6801001294	395397522	5/28/2000
9450885148	365266496	4/3/2005
8788752162	410992357	6/27/2003
6772599870	939331477	11/17/2007
2706093611	118133299	2/13/2012
6080354105	830698187	2/3/2005
7962363951	377755264	12/28/2009
3989382938	481774668	5/23/2013
7871087340	360469706	1/27/2010
9498336898	587626915	7/24/2003
4991471332	155547641	12/17/2005
2894730019	811530112	7/23/2010
9700002691	598762099	2/21/2011
7778225775	344071650	8/20/2006
2271739379	505073339	6/1/2015
8377323975	739708086	8/20/2005
5779354249	367253956	5/30/2011
4078860398	214503975	2/22/2003
5870316974	922396748	11/10/2000
6193585148	809506734	11/11/2007
3302485838	676035108	10/29/2003
2947633522	683944458	7/11/2009
5723853359	961560425	3/23/2009
5219485847	161323456	11/26/2011
1220773496	874322532	9/24/2003
1127148240	347255640	6/6/2014
9425488066	232692153	1/26/2004
5269143935	310342556	2/27/2004
3675888416	796603141	3/22/2003
4847717808	728029351	8/13/2008
4250576605	933396958	11/23/2011
6996698034	846829623	6/9/2007
4725486781	767307813	1/23/2008
8087860788	143395129	5/29/2014
4419136851	456248069	1/5/2004
5012229530	693884384	11/30/2008
9775831377	715270692	3/3/2003
1409024896	230479719	3/3/2003
7861341282	249222518	7/1/2016
8997358153	851596148	8/14/2012
1138497509	540772127	6/30/2014
9206835752	932140639	7/8/2015
7898220177	483306669	10/4/2006
5797121228	957567205	4/1/2007
4504511539	585228551	10/11/2011
2335546298	803702538	11/11/2000
9674152809	239854209	8/25/2012
8022872025	718993588	11/27/2003
2560030289	520443381	12/5/2003
7344796621	809343622	5/15/2014
4219938981	894353514	5/9/2001
7285938521	640427867	7/29/2013
3417929732	517575486	11/14/2007
5704956159	841898675	6/24/2007
6131424738	973798390	7/16/2003
2761621420	410691997	6/20/2013
7591495973	918769106	10/4/2005
1381568267	504140821	2/15/2008
3661132032	198442622	6/14/2000

Attending

Attending	
StudentID	SchoolID
1623605894	213411904
9947971888	818439985
1659748489	699495256
6749495814	230702781
6732746347	525756160
2907244431	853321437
7923860835	203928747
4833465687	440606684
1201487744	316569229
3676340272	933366030
9654624096	570163995
8452734920	160493913
1211640097	901284857
3162539834	902221958
6928053367	380320361
9253525449	866670395
5200269721	795342465
9535197197	797658670
4091224825	881905211
6713690105	794589367
7119805088	895783056
8110245873	735276059
2224209046	160606970
4324466366	595114830
6519298230	208644749
1272128338	328966630
6794117149	618310695
3598466041	390233841
5777880653	782227048
6707774603	253794867
4323506824	312277356
2661745265	513399525
3977299436	574218578
5500881970	550211024
3921917500	881600610
7693076576	712695093
1345145222	752387207
6475963752	564754885
4242628961	654087096
8482406871	527125716
8509647535	278114191
2305306867	989225188
2385025131	125788323
2304300038	217423801
3457268134	441565092
6627918869	304780640
7468175721	138655580
1899353211	979670504
8416133805	905074817
2483863045	978676670
3445846421	324356700
7711034593	961055423
7454387378	386869447
4072188177	532505783
8254400374	875050433
3992856834	782277054
7226972584	646302304
3554663060	787336767
7793889260	677341790
1969177545	188137019
8510679454	885648945
7961279541	478924343
7974251254	744401669
8313792866	539999736
7588813289	457614484
6450244121	124037483
8384688095	811896807
4197289443	484226351
3224420349	254798622
3032303030	581168779
8653375064	962270095
4294114606	904469787
8209244819	526463564
4680996962	116188854
2388561927	385119020
9976610814	314242501
2476390627	370528783
2552205268	486002508
6287820324	487617262
2786418663	321445003
4452512900	778152475
3642945841	584707894
7724186270	749143918
2228763089	204056419
4876820615	732301057
6947552346	636372944
6560214255	707559063
7065997673	382669799
2127573177	840089475
6180571542	959332367
8433977999	807763087
7177329479	576162680
8596655421	435237617
8488457862	594923935
8722034492	829256981
9447702857	919687495
3556652046	694327219
1482827632	220204810
8463376036	252583514
1813454103	940817381

2.4. SAMPLE QUERIES TO YOUR DATABASE

In this section we will describe and show how to retrieve Data from our Organization's Database with useful Queries. These Queries will be written in mathematical expressions that are used for Querying Relational Databases.

2.4.1 DESIGN OF QUERIES

The following sections will describe the three formal Query languages discussed in class and in our textbook. Those languages are: ***Relational Algebra, Tuple Relational Calculus, and Domain Relational Calculus***. Each language description will be followed by sample queries in that language with Relations from our Organization's Database.

2.4.2 RELATIONAL ALGEBRA EXPRESSIONS FOR QUERIES

The ***Relational Algebra*** is a set of operations for retrieving Tuples from a Relational Database state. **Relational Algebra expressions** combine the operations to return sets of tuples. **Relational Algebra expressions** are *Procedural*, this means that they describe a process for retrieving the Tuples from a Relational Database, so the order and nesting of **Relational Algebra expressions** is important.

1. List Students who received a Financial Aid Package of under 1,000 and whose parents are Taxpayers.
   S: Student D:Distribute To **F**: Financial Aid Package **U**: Uses Info From **P**:Parent

π_{S*} [S ✳ $\pi_{U.StudentID.*}$ [$\pi_{D.*}$ (D $\bowtie_{(D.faPackageID = F.packageID)\wedge(F.Amount<1,000)}$ F) $\bowtie_{(D.StudentID = U.StudentID)}$ $\pi_{U.StudentID}$($\pi_{U.*}$(S $\bowtie_{(S.StudentID = U.StudentID)}$ U) $\bowtie_{(U.ParentSsn = P.ssn)\wedge(P.Status=\text{"Taxpayer"})}$ P)]]

---

2. List financial aid packages which contained the same amount for every student.
   FA: Financial Aid Package **D:** Distributed To

$\pi_{(fa1.studentID,\ fa1.Amount)}(FA^{fa1} \bowtie D^{D1}) \div \pi_{(fa2.Amount)}(FA^{fa2} \bowtie_{(fa2.packageID = d2.faPackageID)} D^{D2})$

---

3. List schools visited to by Employees whose hire date is before 1992
   S: School **E:** Employee **O:** Outreach

   $\pi_{S.*}$[S $\bowtie_{(S.DepartmentID = O.DepartmentID)}$ ($\pi_{D.*}$(E$\bowtie_{(E.DepartmentID = O.DepartmentID)\wedge(E.hire_date < 1992)}$O))]

---

4. Find the most expensive budget in the history of the organization.
   B: Budget

   $\pi_{B.*}(B - \pi_{b2.*}(B^{b1} \bowtie_{(b1.Amount > b2.Amount) \wedge (b1.BudgetID != b2.BudgetID)} B^{b2}))$

---

5. Find the second most expensive budget in the history of organization.
   B:Budget **2ME:** 2nd Most Expensive Budget

   $Result \leftarrow B^{b3} - \pi_{B.*}(B - \pi_{b2.*}(B^{b1} \bowtie_{(b1.Amount > b2.Amount) \wedge (b1.BudgetID != b2.BudgetID)} B^{b2}))$
   $2ME \leftarrow \pi_{R1.*}(Result^{R1} \bowtie_{(R1.Amount > R2.Amount) \wedge (R1.BudgetID != R.BudgetID)} Result^{R2})$

---

6. List the academic standing of ALL the students that did NOT receive Financial Aid (All the BAD STUDENTS).
   S: Student **D:** Distributed To

   $\pi_{S.AcademicStanding}[S - \pi_{s1.*}(S^{s1} \bowtie_{s1.StudentID = D.StudentID} D)]$

---

7. Find the schools which have ALL of their students applications approved.
   S: Student **C:** School **F:** Fills Out **A:** Attending **P:** Application

   $C * [\pi_{c.schoolID, s.studentID}(C * A * S) \div \pi_{s.studentID}(S * F * (\sigma_{p.approved = true} P))]$

---

8. Find the school in which the student gets the highest award.
   C: School **S:** Student **A:** Attending **D:**Distributed **F:** Financial Aid Package

   $\pi_{s1.studentID} [(S1 * A * C) * \pi_{s.studentID} [(F3 - (F \bowtie_{(f.amount < f2.amount)} F2)) * D * S]]$

---

9. Find the school in which ALL the students are in good academic standing (All GOOD STUDENTS)
   S:Student **C:**School **A:** Attending

   $C * \pi_{A.SchoolID}[A \div \pi_{s.*}(\sigma_{s1.AcademicStanding='Good'}(Student^{s1})]$

---

10. List students who filled out an Application on each of ALL the days as students named John Doe did on January 02,1992.
    S: Student **F:** Fills Out

    $\pi_{s.*} [S^{s2} * [\pi_{f.StudentID, f.Date}(F) \div (\pi_{f.Date}(S^{s1} \bowtie_{(s1.fName = 'John' \wedge s1.lName = 'Doe' \wedge F.date = 01/02/1992)} F))]]$

2.4.3 TUPLE RELATIONAL CALCULUS EXPRESSIONS FOR QUERIES

Tuple Relational Calculus is a querying language that is nonprocedural, but declarative. **Tuple Relational Calculus expressions** describe the set of Tuples that will be retrieved. These expressions make use of **free variables**, which describe what the query will retrieve, and **bound variables**, which are bounded by *Universal Quantifiers* **(∀)** or *Existential Quantifiers* ***(∃)***. These expressions also use logical expressions with truth values.

1. List Students who received a Financial Aid Package of under 1,000 and whose parents are Taxpayers.

{s|Student(s) ^ ($\exists_d$)($\exists_f$)($\exists_u$)($\exists_p$)[Distributed To(d) ^ FinancialAidPackage(f) ^ Uses info From(u) ^ Parent(p) ^ d.faPackageId = f.PackageID ^ f.amount < 1,000 ^ d.StudentID = u.StudentID ^ u.StudentID = s.StudentID ^ u.PSsn = p.Ssn ^ p.Status = 'Taxpayer']}

---

2. List financial aid packages which contained the same amount for every student.

{f|Financial Aid Package(f) ^ ($\forall_d$)[Distributed To(d) ^ ($\exists_{f2}$) (Financial Aid Package(f2) ^($\exists_{d2}$)(DistributedTo(d2) ^ f2.packageID = d2.faPackageID)) → f.Amount = f2.Amount ^ f.packageID = d.faPackageID]}

---

3. List schools visited to by Employees whose hire date is before 1992

{s|School(s) ^ ($\exists_e$)(Employee(e) ^ ($\exists_o$) (Outreach(o) ^ e.DepartmentID = O.DepartmentID ^ e.hire_date < 01/01/1992 ^ S.DepartmentID = O.DepartmentID))}

---

4. Find the most expensive budget in the history of the organization.

{b|Budget(b) ^ ($\forall_{b2}$)(Budget(b2) ^ b2.Amount < b.Amount →¬($\exists_{b3}$)(Budget(b3) ^ b3.Amount > b.Amount))}

---

5. Find the second most expensive budget in the history of organization.

{b|Budget(b) ^ ($\forall_{b2}$)[Budget(b2) ^ b.Amount > b2.Amount →($\exists_{b3}$)(Budget(b3) ^ b3.Amount > b.Amount ^b2.BudgetID != b.BudgetID)]}

---

6. List the academic standing of ALL the students that did NOT receive Financial Aid (All the BAD STUDENTS).

{n'|($\exists_s$)Student(s) ^ n'.AcademicStanding = s.AcademicStanding ^ ($\forall_d$) [DistributedTo(d) → s.StudentID != d.studentID]

---

7. Find the schools which have ALL of their students applications approved.
S: Student **C:** School **F:** Fills Out **A:** Attending **P:** Application

{c|School(c) ^ ($\forall_s$)[Student(s) ^ ($\exists_p$)(Application(p) ^ ($\exists_f$)(Fills Out(f) ^ f.ApplicationID = p.ApplicationID ^ p.Approved = True)) → ($\exists_a$)(Attending(a) ^ a.StudentId = S.StudentID ^ a.SchoolID = C.SchoolID)]}

---

8. Find the school in which the student gets the highest award.
C: School **S:** Student **A:** Attending **D:**Distributed **F:** Financial Aid Package

{c | school(c) ^ ($\exists_s$)($\exists_f$)($\exists_d$)($\exists_a$)($\forall_{f2}$)(student(s) ^ Financial Aid Package(f) ^ Financial Aid Package(f2) ^ Distributed To(d) ^ Attending(a) ^ (f.fid != f2.fid) ^ d.fid = f.fid ^ d.sid = s.sid → f.amount > f2.amount ^ c.schoolID = a.schoolID^ a.studentId = s.studentID)}

---

9. Find the school in which ALL the students are in good academic standing (All GOOD STUDENTS)
C: School **S:** Student **A:** Attending

{c|School(c) ^ ($\forall_s$)[Student(s) → ($\exists_a$)(Attending(a) ^ s.StudentID = a.StudentID ^ a.SchoolID = c.SchoolID ^ s.AcademicStanding = 'Good')]}

---

10. List students who filled out an Application on each of ALL the days as students named John Doe did on January 02,1992.

{s|Student(s) ^($\forall_f$)[Fills Out(f) ^ f.date = 01/02/1992 ^($\exists_{fa2}$)(Fills Out(fa2) ^ fa2.date = f.date ^ ($\exists_{sj}$)(Student(sj) ^ sj.fName ='John' ^ sj.lName = 'Doe' ^ sj.studentID = fa2.StudentID)) → s.studentID = f.studentID]}

2.4.4 DOMAIN RELATIONAL CALCULUS EXPRESSIONS FOR QUERIES

The ***Domain Relational Calculus*** is a variation of Relational Calculus. In the ***Domain Relational Calculus,*** each variable represents a single value within a Tuple, instead of a Tuple itself.

1. List Students who received a Financial Aid Package of under 1,000 and whose parents are Taxpayers.
   Student, Distributed To, FA Package, Parent, Uses info From

{<s>|Student(s,_,_,_,_,_,_,_,_,_,_,_,_,_) ^ Financial Aid Package(f, _,<1,000,_) ^ Distributed To(s, f,_) ^ ($\exists_{p}$)(Parent(p,_,_,_,_,_,_,_,_,_,_,'Taxpayer',_) ^ Uses info from(s, p)) }

---

2. List financial aid packages which contained the same amount for every student.

{<f,t,a,b>|Financial Aid Package(f,t,a,b) ^ ($\forall_{s}$)[Distributed To(s,f,_)
→ ($\exists_{f2}$)($\exists_{t2}$)(Financial Aid Package(f2,t2,a,_) ^ Distributed To(s, f2,_))]}

---

3. List schools visited to by Employees whose hire date is before 1992

{<s, d>|School(s,_,_,_,_,_,d) ^ ($\exists_{e}$) (Employee(e,_,<01/01/1992,_,_,_,_,_,_,_,_,_,d) ^ Outreach(d)}

---

4. Find the most expensive budget in the history of the organization.

{<b,a,d>|Budget(b,a,d) ^ ($\forall_{b2}$)[Budget(b2,<a,_) → ¬($\exists_{b3}$) (Budget(b3 ,>a,))]}

---

5. Find the second most expensive budget in the history of organization.

{<b,a,d>|Budget(b,a,d) ^ ($\forall_{b2}$)[Budget(b2,<a,_) → ($\exists_{b3}$)(Budget(b3,>a,))]}

---

6. List the academic standing of ALL the students that did NOT receive Financial Aid (All the BAD STUDENTS).

{<s,a> | Student (s,_,a,_,_,_,_,_,_,_,_,_,_,_) ^ ($\forall_{d}$) [Distributed To (d,_,_) → Student(d,_,_,_,_,_,_,_,_,_,_,_,_,_)]}

---

7. Find the schools which have ALL of their students applications approved.

{<c>|School(c,_,_,_,_,_,) ^ ($\forall_{s}$)[Students(s,_,_,_,_,_,_,_,_,_,_,_,_,_) ^ ($\exists_{p}$)(Application(p,_,_,_,True) ^ Fills Out(s,p,_)) → Attending(s, c)]}

---

8. Find the school in which the student gets the highest award.

{<c>|School(c,_,_,_,_,_) ^ ($\exists_s$)[Student(s,_,_,_,_,_,_,_,_,_,_,_,_,_) ^($\exists_{f1}$) ($\exists_a$) (Financial Aid Package(f1,_,a,_) ^ ($\exists_{f2}$)(Financial Aid Package(f2,_,<a,_) ^ Distributed To(s,f1,_) ^ Attending(s, c)]}

---

9. Find the school in which ALL the students are in good academic standing (All GOOD STUDENTS)

{<c>|School(c,_,_,_,_,_) ^ ($\forall_s$)[Students(s, _, 'Good', _, _, _, _, _, _, _, _, _, _, _) → Attending(s, c)}

---

10. List students who filled out an Application on each of ALL the days as students named John Doe did on January 02, 1992.

{<s>|Student(s, _, _, _, _, _, _, _, _, _, _, _, _, _) ^ ($\forall_f$) [Fills Out(s,f,01/02/1992) → ($\exists_{f2}$) ($\exists_{s2}$) (Fills Out(s2, f2, 01/02/1992) ^ Student(s2, _, _, 'John', _, 'Doe', _, _, _, _, _, _, _, _))]}

[This page intentionally left blank]

Phase Three:
Oracle Database Management System

3.1 NORMALIZATION OF RELATIONS

In the previous section, we mapped our Conceptual E-R Model to a Logical Relational Model. Additionally, we introduced mock data as well as Queries in three mathematical Query languages. In order to implement the Logical Database into a physical database, we must analyze and critique the design of our Relational Database Schema.

This section introduces a formal method for measuring the design of a Relational Database Schema. That method is called ***Normalization***. We will also describe the problems that occur when operating on a poorly normalized database design. Finally, our *California Aid* relation schemas will be analyzed.

3.1.1 NORMALIZATION AND NORMAL FORMS

DESCRIPTION OF NORMALIZATION AND NORMAL FORMS

Normalization can be described as a process of analyzing relation schemas in order to minimize redundancy as well as minimizing anomalies.
Normalization provides a formal framework for analyzing Relation Schemas by breaking them apart so that any redundancy is removed and no anomalies occur. All of this is done with a series of ***Normal Form Tests.*** There exists four main *Normal Forms* that a relation schema can satisfy: *First Normal Form(1NF), Second Normal Form(2NF), Third Normal Form(3NF), and Boyce-Codd Normal Form(BCNF)*.

Traditionally, all of the Normal Form tests are followed in a sequence, with the goal of achieving ***3NF*** Relations by progressing through ***1NF*** and ***2NF***. The following is a description of each Normal Form.

First Normal Form (1NF)

The First Normal Form (***1NF***) states that the domain of an attribute must *only* include *atomic values* (simple and indivisible) and the value of any attribute of a Tuple must be a *single value* from the domain of the attribute. This means that ***1NF*** does not allow having a set of values, a tuple of values, or a combination of both as an attribute value for a single Tuple.

A Relation Schema that is not ***1NF*** can be fixed multiple ways:
1. Multi-valued attributes can be made into a separate Relation that contains the original Relation's Primary Key as a Foreign Key attribute. This is the same method that was used to map multi-value attributes in the Conceptual E-R Model to Relations in Phase 2.

2. If the multi-valued attribute contains a specific number *N* of values for each tuple, then a new single-value attribute is added to the Relation Schema for each N_i.

3. A multi-valued attribute will be replaced with a new single-value attribute, and for each of the original multiple values a new tuple is duplicated. This means that if a Tuple *A* has a multi-valued attribute m_3, then *A* will be duplicated and each m_i will be a single-value attribute to each A_i. This method results in duplicating all the other attributes, so it is not a good approach.

Second Normal Form (2NF)

The Second Normal Form (***2NF***) is based on the concept of *Full Functional Dependency*. In order to understand a *Full Functional Dependency*, *Functional Dependencies* must be described.

Functional Dependency: Denoted by $X \rightarrow Y$
In a Tuple, a set of attributes *Y* is *functionally dependent* on another set of attributes *X* if the set of values for *X* map to only one set of values for *Y*.

This means that the values of *X* can be used to determine the values of *Y*. For example, a Tuple's Primary Key value maps to only one Tuple.

1. A Relation Schema satisfies ***2NF*** if the Relation Schema satisfies ***1NF***.

2. All attributes that are not part of the Primary Key must *fully functionally depend* on the Primary Key. *Fully functionally dependence* means that once one of the attributes of a Primary Key is removed, the functional dependency no longer holds. This applies to Relation Schema that have more than one attribute as their primary key.

3. If a Relation Schema Primary Key only has a single attribute, it automatically passes the ***2NF*** test.

4. A Relation Schema that fails the ***2NF*** test can be normalized by breaking it down into smaller Relation Schemas. The Primary Keys of these smaller Relation Schemas will be subsets of the original Primary Key.

Third Normal Form (3NF)

The Third Normal Form (***3NF***) is based on the concept of *transitive dependency*. In a Relation Schema *R*, a functional dependency *X* → *Y* is *transitive dependency* if there exists a set of attributes (*A*) in *R* that is neither a Candidate Key nor a subset of any Key of *R*. *A* will also functionally depend on *X*, and *Y* will also functionally depend on *A* (*X* → *A* and *A* → *Y* must also be true).

1. A Relation Schema satisfies ***3NF*** if it satisfies ***2NF*** and ***1NF***.

2. A Relation Schema satisfies ***3NF*** when there does not exist any *non-prime* (not part of Primary Key) *attributes* that functionally depend on other *non-prime attributes*. This is the *transitive dependency* mentioned earlier.

3. A Relation Schema that fails the ***3NF*** test can be normalized by being broken down into Relations where the left side of a functional dependency is always a Primary Key attribute (or Superkey which contains the Primary Key).

Boyce-Codd Normal Form (BCNF)

Boyce-Codd Normal Forms is considered a simpler form of ***3NF***, yet it is stricter than ***3NF***. Therefore, it is justified to say that every Relation Schema that is ***BCNF*** is also ***3NF***. On the other hand, a Relation Schema that is ***3NF*** is not necessarily ***BCNF***. The reason why ***BCNF*** is stricter than ***3NF*** is because it does not allow any prime attributes (members of primary or candidate keys) to depend on non-prime attributes.

1. A Relation Schema satisfies ***BCNF*** when all previous normal forms (***1NF 2NF 3NF***) are satisfied.
2. The left side of any functional dependency must be a Primary key (or Superkey) of the Relation Schema.
3. A Relation Schema that fails the ***BCNF*** test can be broken down into Relations where non-prime attributes at the left side of any functional dependency become prime attributes of the new Relation Schemas.

ANOMALIES THAT RESULT FROM POOR NORMALIZATION

Poor normalization of data can result in three classes of anomalies: ***insertion***, ***modification***, and ***deletion***.

Insertion Anomalies

Storing Natural Joins of base Relations lead to Insertion Anomalies. These anomalies can be described in two ways.

First:
Before inserting two Tuples that represent two Relations which are both Joined to the same Relation, the Attribute values of the Joined Relation must be *exactly* the same for both Tuples in order for the data to be *coherent*.

Second:
Before inserting a Tuple representing a Relation that is not Joined to any other Relation, attribute values for the other Relation Schema must all be set to NULL.

This is problematic because NULL values don't have a single interpretation. If any of the attributes that are set to NULL help compose the Primary Key of the Joined Relation, then the Entity Integrity Constraint will be violated as we stated in Phase 2.

Modification Anomalies

Attribute values representing a single Relation appear in all Tuples if a set of Tuples can represent one single Relation that is Joined to several other Relations. If any of those attribute values are changed into one Tuple, they must be changed for all Tuples in order for the data to maintain its coherency.

Deletion Anomalies

If a set of Tuples can represent a Relation that is Joined to other Relations, and the Tuples are removed, then any Record of the single Relation will be completely removed from the Database. This will result in the single Relation not being Joinable to other Relations.

3.1.2 NORMAL FORMS FOR OUR DATABASE

In this section, we will check our Relation Schemas to determine if they satisfy at least ***3NF***. Afterword, we will create Relation Schema to discuss the anomalies that can occur for such a Relation Schema.

Employee

Functional Dependencies:
FD1 {**employeeID**} → {SSN, Hire Date, End Date, fName, …, DepartmentID}
FD2 {SSN} → {**employeeID**, fName, mName,…DepartmentID}
Candidate Keys:
employeeID (Primary Key)
SSN
Normal Forms:
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Student

Functional Dependencies:
FD1 {**StudentID**} → {SSN, Academic Standing, …, Income Status}
FD2 {SSN} → {**StudentID**, Academic Standing, …, Income Status}
Candidate Keys:
StudentID (Primary Key)
SSN
Normal Forms:
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Parent

Functional Dependencies:
FD1 {**SSN**} → {fName, mName, lName, …, Status, Income}
Candidate Keys:
SSN (Primary Key)
Normal Forms:
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Department

<u>Functional Dependencies:</u>
FD1 {**DepartmentID**} → {Name}
<u>Candidate Keys:</u>
DepartmentID (Primary Key)
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Logistics

<u>Functional Dependencies:</u>
No functional dependencies exist.
<u>Candidate Keys:</u>
DepartmentID (Primary Key)
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

Outreach

<u>Functional Dependencies:</u>
No functional dependencies exist.
<u>Candidate Keys:</u>
DepartmentID (Primary Key)
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

Review Board

<u>Functional Dependencies:</u>
FD1 {**DepartmentID**} → {BudgetID}
<u>Candidate Keys:</u>
DepartmentID (Primary Key)
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

School

Functional Dependencies:

FD1 {**SchoolID**} → {Name, Street, City, …, DepartmentID}

Candidate Keys:

SchoolID (Primary Key)

Normal Forms:

1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Application

Functional Dependencies:

FD1 {**AppID**} → {requestedAmount, Date, Status, Approved}

Candidate Keys:

AppID (Primary Key)

Normal Forms:

1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Data

Functional Dependencies:

FD1 {**DataID**} → {Description, Date, DepartmentID}

Candidate Keys:

DataID (Primary Key)

Normal Forms:

1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Information

Functional Dependencies:

FD1 {**SourceID**} → {Date, DepartmentID}

Candidate Keys:

SourceID (Primary Key)

Normal Forms:

1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Budget

<u>Functional Dependencies:</u>
FD1 {**BudgetID**} → {Amount, Date}
<u>Candidate Keys:</u>
BudgetID (Primary Key)
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Financial Aid Package

<u>Functional Dependencies:</u>
FD1 {**packageID**} → {Type, Amount, BudgetID}
<u>Candidate Keys:</u>
packageID (Primary Key)
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Uses info from

<u>Functional Dependencies:</u>
FD1 {**StudentID**} → {**ParentSsn**}
FD1 {**ParentSsn**} → {**StudentID**}
<u>Candidate Keys:</u>
StudentID, ParentSsn (Primary Key)
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

Fills Out

<u>Functional Dependencies:</u>
FD1 {**StudentID**} → {**ApplicationID**,Date}
FD2 {**ApplicationID**} → {**StudentID**, Date}
<u>Candidate Keys:</u>
StudentID (Primary Key)
ApplicationID
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Becomes

<u>Functional Dependencies:</u>
FD1 {**AppID**} → {**DataID, DataDescription,** Date}
FD2 {**DataID**} → {**AppID,DataDescription**, Date}
<u>Candidate Keys:</u>
AppID (Primary Key)
DataID, DataDescription
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Uses and Stores Into

<u>Functional Dependencies:</u>
FD1 {**DataID**} → {**DataDescription, SourceID,** SourceDate}
FD2 {**DataDescription**} → {**DataID, SourceID**, SourceDate}
FD3 {**SourceID**} → {**DataID,DataDescription,** SourceDate}
<u>Candidate Keys:</u>
DataID (Primary Key)
DataDescription
SourceID
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Distributed To

<u>Functional Dependencies:</u>
FD1 {**StudentID**} → {**faPackageID**, Date}
FD2 {**faPackageID**} → {**StudentID,** Date}
<u>Candidate Keys:</u>
StudentID (Primary Key)
faPackageID
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Attending

<u>Functional Dependencies:</u>
FD1 {**StudentID**} → {**SchoolID**}
FD2 {**SchoolID**} → {**StudentID**}
<u>Candidate Keys:</u>
StudentID (Primary Key)
SchoolID
<u>Normal Forms:</u>
1NF is satisfied because all attributes have atomic domains.
2NF is satisfied because the Primary Key only has one attribute.
3NF is satisfied because no non-prime attributes depend on other non-prime attributes.
BCNF is satisfied because the left side of all functional dependencies is a Candidate Key.

Example of Poorly Normalized Relation:

Attending-Information is a Relation Schema created by natural joining the **Attending** and **Information** Relations. We will show its functional dependencies, explain why it does not satisfy all the normal form tests, and illustrate some of the anomalies that occur.
<u>Relation:</u>
attendinginformation(**StudentID**, SchoolID, SourceID, Date, DepartmentID)
<u>Functional Dependencies:</u>
FD1 {**StudentID**} → {SchoolID, SourceID, Date, DepartmentID}
FD2 {SourceID} → {Date, DepartmentID}
<u>Candidate Keys:</u>
StudentID (Primary Key)
<u>Normal Forms:</u>
1NF 2NF are both satisfied because all attributes are atomic and there is only one Primary Key attribute. ***3NF*** is not satisfied because *Date* and *DepartmentID* functionally depend on a non-prime attribute *SourceID*.
<u>Possible Anomalies:</u>
To add a new information source to the database that does not come from a Student, all of the *Attending* fields will have to be NULL.

3.2 SQL *PLUS: MAIN PURPOSE AND FUNCTIONALITY

Now that the relational design of our database has been demonstrated we will discuss the implementation and transition process. The physical database was implemented and loaded with sample data using Oracle SQL Developer program. Additionally, we used SQL * PLUS which is a command-line user interface for interacting with the Oracle DBMS. The main objective of implementing SQL * Plus is to enable database administrators to quickly define and maintain the existing database. It allows users to enter SQL commands to define and manage schema objects, manipulate and query existing data, and control the formatting of output. It also allows users to create and run scripts that execute multiple of the above commands at once. Finally, SQL * Plus allows users to create and run PL/SQL scripts. PL/SQL is Oracle's procedural extension of SQL, combining SQL statements with flow control structures like conditions and loops. PL/SQL programs can be saved as stored procedures and set to automatically run using triggers.

3.3 SCHEMA OBJECTS FOR ORACLE DBMS

Tables

Tables are a basic unit of Storage in an Oracle Database. Data is stored into rows, which retain the attributes of a Relational Schema. In a Table, Columns have names such as *student_id*, *employee_id*, *fName*. Additionally, column names have set widths and specified datatypes such as *NUMBER* and *VARCHAR2*. Once Data is inserted into Tables, that Data can be Updated or queries with SQL.

Syntax:

```
CREATE TABLE [TABLE NAME]
  [column-definition-1],
   ...
  [column-definition-n],
     [table constraints]

[column-definition]:=
   [column-name] [column-datatype] [column-constraints]
[table constraints]:=
CONSTRAINT [constraint-name] PRIMARY KEY [(column-name)],
FOREIGN KEY [(column-name)] REFERENCES
                    [(table-name)] [(column-name)],
UNIQUE [(column-name)], CHECK [boolean-expression]
```

<u>Examples of this Implementation</u>

- EOOO_EMPLOYEE
- EOOO_STUDENT
- EOOO_PARENT
- EOOO_APPLICATION
- EOOO_ATTENDING
- EOOO_BECOMES
- EOOO_BUDGET
- EOOO_DATA
- EOOO_DEPARTMENT
- EOOO_DISTRIBUTEDTO
- EOOO_FAPACKAGE
- EOOO_FILLSOUT
- EOOO_INFOR
- EOOO_LOGISTICS
- EOOO_REVIEWBOARD
- EOOO_OUTREACH
- EOOO_SCHOOL
- EOOO_USTORESI
- EOOO_UINFOF

Views

Views are the result of a query stored as Virtual Tables. This means they do not store data, but a SELECT statement that generates a specific representation of the Data instead. This way, a Database Administrator can control which data is available, how it is presented, and how it is formatted. Front-end applications tend to retrieve queries from views instead of tables because it saves time and simplifies queries. SELECT, CREATE, INSERT, UPDATE, DELETE all apply to views like they apply to Tables. Views are dynamically created when a base Table is updated.

Syntax:

```
CREATE [OR REPLACE] VIEW view_name
 AS
    [select statement/query]
```

Procedures *implemented in phase 4*

Procedures are stored blocks of PL/SQL code that can be run via the command line or in scripts Because procedures are stored, they are reusable. Procedures take advantage of the flow control structures provided by PL/SQL. This means that they can make more complex operations that pure SQL.

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
BEGIN
      [PL/SQL Statement]
END
```

Triggers

Triggers are stored blocks of PL/SQL code that automatically run each time a specific event occurs (like an INSERT)

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
AFTER
        [event_name]
ON
       [table_name]
BEGIN
      [PL/SQL statement]
END
```

Packages

Packages are groupings of PL/SQL objects and procedures that provide an interface to more complication SQL * Plus functionality. They abstract and encapsulate data and functions similarly to Classes in Object-Oriented Programming (OOP). Packages contain a spec block that defines the public interface to the package, as well as a body block which fully defines the hidden code within procedures. Packages allow for more complex procedures to be reused easily and kept hidden from front-end developers.

Syntax:

```
CREATE [OR REPLACE] PACKAGE package_name
AS
   [object and procedure declarations]
END
CREATE [OR REPLACE] PACKAGE BODY package_name
AS
   [object and procedure declarations]
END
```

Sequence Generators

Sequence generators use a mathematical function to produce a sequence of unique values. Each time the sequence generator is requested, it responds with the next number in the sequence. Sequence generators are often used to generate unique values for Primary Key attributes, and ensure that unique Primary Key values are used for new Tuples being inserted by multiple users at the same time. Sequence generators have a caching option which allows the generator to pre-calculate and store the next *n* numbers in the sequence in memory.

Syntax:

```
CREATE SEQUENCE sequence_name
MINVALUE minimum_value
MAXVALUE maximum_value
START WITH starting_number
INCREMENT BY increment_size
CACHE cache_size
```

Indexes

Indexes serve the purpose of providing faster access paths to specified table columns which speed up queries. Columns may be used in multiple indexes if each index contains a unique set of columns. Oracle automatically creates indexes for Primary Keys. Indexes are logically and physically independent as they may be created and dropped at any time without affecting the Table Data or other indexes. Oracle provides the following indexing schemes which correspond to speed improvements.

- B-tree indexes
- B-tree cluster indexes
- Hash cluster indexes
- Reverse Key indexes
- Bitmap indexes
- Bitmap join indexes

3.4 LIST RELATIONS WITH SQL COMMANDS

Application:

```
CS3420 SQL> desc eooo_application
Name                   Null?       Type
-------------------- ----------- ----------
APPLICATIONID          NOT NULL    NUMBER(9)
REQUESTEDAMOUNT        NOT NULL    FLOAT(10)
APPDATE                NOT NULL    DATE
APSTATUS               NOT NULL    VARCHAR2(30)
APPROVED               NOT NULL    VARCHAR2(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_application a where a.apid < 20;
APID  RAMOUNT APPDATE   APSTATUS   APPROVED
---- -------- --------- ---------- ----------
   1    91210 18-MAY-09 complete   t
   2   526900 02-JUN-09 incomplete f
   3   401100 08-APR-97 complete   t
   4   739600 15-SEP-96 incomplete f
   5   315300 22-JUL-04 complete   t
   6   536400 30-NOV-04 incomplete f
   7   864100 15-MAY-09 complete   t
   8   320400 04-NOV-94 incomplete f
   9    54600 20-AUG-05 complete   t
  10   186200 22-JAN-97 incomplete f
  11   205700 01-AUG-14 complete   t
  12   983800 15-NOV-12 incomplete f
  13   949100 06-MAY-05 complete   t
  14   904200 21-JUL-03 incomplete f
  15   230500 21-APR-12 complete   t
  16    88580 21-DEC-04 incomplete f
  17   873500 23-NOV-14 complete   t
  18   987100 05-JAN-00 incomplete f
  19    25630 22-JUN-00 complete   t
  20   585200 20-AUG-05 incomplete f

20 rows selected.
```

Attending:

```
CS3420 SQL> desc eooo_attending
Name             Null?    Type
---------------- -------- ---------
STUDENTID        NOT NULL NUMBER(9)
SCHOOLID         NOT NULL NUMBER(9)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_attending a where a.schoolid < 20;
STUDENTID   SCHOOLID
---------- ----------
         1          1
         2          2
         3          3
         4          4
         5          5
         6          6
         7          7
         8          8
         9          9
        10         10
        11         11
        12         12
        13         13
        14         14
        15         15
        16         16
        17         17
        18         18
        19         19
        20         20

20 rows selected.
```

Becomes:

```
CS3420 SQL> desc eooo_becomes
Name              Null?    Type
----------------- -------- ------------
APPID             NOT NULL NUMBER(9)
DATAID            NOT NULL NUMBER(9)
DATADESCRIPTION   NOT NULL VARCHAR2(50)
BDATE             NOT NULL DATE

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_becomes b where b.appid < 20;
APPID       DATAID     DATADESCRIPTION          BDATE
----------- --------- ----------------------- ---------
         1          1 parent                  21-MAY-15
         2          2 student                 16-JUN-83
         3          3 parent                  27-APR-16
         4          4 student                 31-JUL-95
         5          5 parent                  02-MAR-09
         6          6 student                 09-NOV-85
         7          7 parent                  20-NOV-88
         8          8 student                 24-FEB-89
         9          9 parent                  11-APR-86
        10         10 student                 04-APR-16
        11         11 parent                  16-MAY-15
        12         12 student                 29-NOV-82
        13         13 parent                  21-APR-12
        14         14 student                 17-APR-90
        15         15 parent                  11-FEB-84
        16         16 student                 25-MAY-80
        17         17 parent                  03-JAN-91
        18         18 student                 20-JUL-94
        19         19 parent                  27-APR-83
        20         20 student                 26-NOV-94

20 rows selected.
```

Budget:

```
CS3420 SQL> desc eooo_budget
Name                 Null?    Type
---------------- -------- ----------
BUDGETID            NOT NULL NUMBER(9)
AMOUNT              NOT NULL FLOAT(15)
BUDATE              NOT NULL DATE

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_budget b where b.budgetid < 30;
  BUDGETID     AMOUNT BUDATE
---------- ---------- ---------
         1     420230 13-MAY-02
         2     248660 18-SEP-15
         3     702670 10-MAR-08
         4     137350 28-JUN-91
         5     518440 02-JUN-15
         6     794010 06-SEP-89
         7     512830 16-JAN-03
         8     373130 19-SEP-88
         9     500940 15-OCT-13
        10     968300 22-JUN-99
        11     725520 17-JAN-93
        12     472420 19-NOV-86
        13     969120 12-JAN-14
        14      90121 19-APR-07
        15     469580 10-JUL-87
        16     384370 04-OCT-11
        17     148800 15-MAR-95
        18     989340 30-AUG-05
        19     999090 05-JAN-88
        20     100890 18-OCT-91
        21     450920 16-MAY-06
        22     819250 16-JAN-06
        23      71253 02-MAR-11
        24     144230 11-FEB-89
        25     223810 13-JUN-96
        26     870420 29-APR-98
        27     295030 06-JAN-14
        28     740480 08-NOV-88
        29     851730 10-SEP-99
        30     146200 09-JAN-83

30 rows selected.
```

Data:

```
CS3420 SQL> desc eooo_data
Name             Null?    Type
---------------- -------- --------------
DATAID           NOT NULL NUMBER(9)
DESCRIPTION      NOT NULL VARCHAR2(30)
DATADATE         NOT NULL DATE
DEPARTMENTID     NOT NULL NUMBER(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_data d where d.dataid < 20;
    DATAID DESCRIPTION                    DATADATE  DEPARTMENTID
---------- ------------------------------ --------- ------------
         1 parent                         21-AUG-82            2
         2 student                        30-DEC-01            2
         3 parent                         10-JUN-90            2
         4 student                        25-SEP-15            2
         5 parent                         04-MAY-11            2
         6 student                        18-DEC-03            2
         7 parent                         18-JUN-85            2
         8 student                        23-JUN-12            2
         9 parent                         02-OCT-86            2
        10 student                        22-OCT-06            2
        11 parent                         23-FEB-83            2
        12 student                        22-AUG-86            2
        13 parent                         27-APR-01            2
        14 student                        05-MAY-10            2
        15 parent                         04-JUN-89            2
        16 student                        24-MAY-99            2
        17 parent                         01-FEB-97            2
        18 student                        07-FEB-98            2
        19 parent                         02-FEB-86            2
        20 student                        11-FEB-85            2

20 rows selected.
```

Department:

```
CS3420 SQL> desc eooo_department
Name             Null?    Type
---------------- -------- -------------
DEPARTMENTID     NOT NULL NUMBER(1)
NAME             NOT NULL VARCHAR2(30)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_department;
DEPARTMENTID NAME
------------ --------------------
           2 Logistics
           4 Outreach
           6 Review Board
```

Distributed To:

```
CS3420 SQL> desc eooo_distributedto
Name             Null?    Type
---------------- -------- ----------
STUDENTID        NOT NULL NUMBER(9)
FAPACKAGEID      NOT NULL NUMBER(9)
DDATE            NOT NULL DATE

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_distributedto d where d.studid < 20;
STUDID     FAPACKAGEID DDATE
---------- ----------- ---------
         1           1 06-OCT-03
         2           2 04-SEP-00
         3           3 19-NOV-12
         4           4 07-MAR-02
         5           5 11-MAR-09
         6           6 10-JUL-00
         7           7 27-MAY-96
         8           8 16-JUN-06
         9           9 24-DEC-09
        10          10 19-SEP-96
        11          11 31-OCT-91
        12          12 28-AUG-11
        13          13 20-MAR-00
        14          14 01-DEC-01
        15          15 11-MAY-09
        16          16 02-OCT-97
        17          17 14-AUG-97
        18          18 03-JUL-00
        19          19 03-FEB-97
        20          20 14-DEC-13

20 rows selected.
```

Employee:

```
CS3420 SQL> desc eooo_employee
Name               Null?    Type
---------------- -------- ------------
EMPLOYEE_ID        NOT NULL NUMBER(10)
SSN                NOT NULL NUMBER(9)
HIRE_DATE          NOT NULL DATE
END_DATE           NOT NULL DATE
FNAME              NOT NULL VARCHAR2(30)
MNAME                  NULL VARCHAR2(30)
LNAME              NOT NULL VARCHAR2(30)
STREET             NOT NULL VARCHAR2(50)
CITY               NOT NULL VARCHAR2(30)
STATE              NOT NULL VARCHAR2(30)
ZIP                NOT NULL NUMBER(5)
PHONE_NUMBER       NOT NULL NUMBER(10)
ESEX               NOT NULL VARCHAR2(1)
DEPARTMENTID       NOT NULL NUMBER(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_employee e where e.eid < 20;
EID        SSN HDATE     EDATE     FNAME   MNAME   LNAME   STR    CITY       STAT    ZIP ES DID
--- ---------- --------- --------- ------- ------- ------- ------ ---------- ---- ------ -- ---
  1  683261326 09-DEC-82 28-MAY-95 Ashley  Pamela  Greene  Glenda Monroe     Cali  56878 f    2
  2  180370469 01-MAR-96 26-MAY-00 Craig   Dennis  Johnsto Los al Chattanoog Cali  57604 m    2
  3  678639536 01-MAR-97 26-MAY-01 Abel    Den     Johns   Stockd Bakersfiel Cali  93301 m    2
  4  360688653 01-MAR-96 26-MAY-00 John    Doe     Doe     Burban Los Angele Cali  59094 m    2
  5  681091061 01-APR-96 26-MAY-00 Jane    Doe     Doe     Cheste Santa Cruz Cali  12342 m    2
  6  122003861 01-APR-96 26-MAY-00 Frank   Mir     Foreman Missio San Franci Cali  99876 m    2
  7  182158566 01-MAY-96 26-MAY-00 Abel    frank   francis cheste Santa Barb Cali  11111 m    2
  8  215106628 01-MAY-96 26-MAY-00 Erik    frankab francis cheste Pismo      Cali  22345 m    2
  9  991116657 01-JUL-96 26-MAY-00 hector  erik    francis H      Santa mari Cali  12323 m    2
 10  991116612 01-JUL-96 26-MAY-00 joe     erika   erik    panama almos      Cali    123 m    2
 11  991116456 01-JUL-96 26-MAY-00 laze    joe     hector  stockd lamont     Cali    123 m    2
 12  991116455 01-JUL-96 26-MAY-00 lacy    laze    lacy    brimha arvin      Cali    123 m    2
 13  991231111 01-JUL-96 26-MAY-00 tasty   lacy    joe     ojes   bakersfiel Cali    123 m    2
 14  991116657 01-JUL-96 26-MAY-00 gordon  tasty   rebeca  tash   burbank    Cali    123 m    2
 15  991122127 01-JUL-96 26-MAY-00 rebeca  gordon  gordon  hash   hollywood  Cali    123 m    2
 16  991122327 01-JUL-96 26-MAY-00 rebeca  coca    nabisco dimlit hollywoho  Cali    123 m    2
 17  991122337 01-JUL-96 26-MAY-00 chris   escobar kelso   lotus  noho       Cali    123 m    2
 18  991122577 01-JUL-96 26-MAY-00 alex    kali    foreman lamar  chesterfie Cali    123 m    2
 19  991122907 01-JUL-96 26-MAY-00 jones   soreen  berkhar smalls chatanoga  Cali    123 m    2
 20  991122917 01-JUL-96 26-MAY-00 smith   friedre pinciat Cresce chatanoga  Cali    123 m    2

20 rows selected.
```

faPackage:

```
CS3420 SQL> desc eooo_fapackage
Name              Null?    Type
---------------- -------- --------------
PACKAGEID         NOT NULL NUMBER(9)
PTYPE             NOT NULL VARCHAR2(30)
AMOUNT            NOT NULL FLOAT(15)
FABUDGETID        NOT NULL NUMBER(9)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_fapackage f where f.pid < 20;
PID        PTYPE                              AMOUNT FABUDGETID
---------- ------------------------------ ---------- ----------
         1 calgranta                          534140          1
         2 calgrantb                          181890          2
         3 other                              718270          3
         4 calgranta                          191930          4
         5 calgrantb                          965840          5
         6 other                              361850          6
         7 calgranta                          988990          7
         8 calgrantb                          229160          8
         9 other                              959560          9
        10 calgranta                          922430         10
        11 calgrantb                          677320         11
        12 other                              503440         12
        13 calgranta                          319310         13
        14 calgrantb                          400700         14
        15 other                              974030         15
        16 calgranta                          180050         16
        17 calgrantb                          846400         17
        18 other                               88847         18
        19 calgranta                          343020         19
        20 calgrantb                          951420         20

20 rows selected.
```

Fills_out:

```
CS3420 SQL> desc eooo_fillsout
Name                 Null?    Type
--------------- -------- ----------
STUDENTID            NOT NULL NUMBER(9)
APPLICATIONID        NOT NULL NUMBER(9)
FDATE                NOT NULL DATE

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_fillsout f where f.studentid < 20;
STUDENTID APPLICATIONID FDATE
---------- ------------ ---------
         1             1 07-DEC-09
         2             2 26-MAY-08
         3             3 09-NOV-89
         4             4 25-DEC-93
         5             5 30-NOV-15
         6             6 14-NOV-91
         7             7 24-JAN-95
         8             8 15-NOV-93
         9             9 26-MAR-15
        10            10 12-APR-13
        11            11 16-JAN-91
        12            12 22-SEP-16
        13            13 12-FEB-10
        14            14 16-OCT-15
        15            15 12-FEB-90
        16            16 28-JUN-92
        17            17 05-JAN-03
        18            18 12-SEP-86
        19            19 13-APR-16
        20            20 31-JUL-01

20 rows selected.
```

Infor:

```
CS3420 SQL> desc eooo_infor
Name                 Null?    Type
--------------- -------- ----------
SOURCEID          NOT NULL NUMBER(9)
INFODATE          NOT NULL DATE
DEPARTMENTID      NOT NULL NUMBER(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_infor i where i.sourceid < 20;
  SOURCEID INFODATE  DEPARTMENTID
---------- --------- ------------
         1 29-AUG-01            6
         2 05-OCT-84            6
         3 10-JUL-09            6
         4 19-APR-93            6
         5 18-JAN-81            6
         6 05-JUL-94            6
         7 11-MAR-04            6
         8 21-AUG-98            6
         9 18-JUL-97            6
        10 18-JUL-83            6
        11 06-SEP-12            6
        12 07-FEB-80            6
        13 03-JUN-01            6
        14 22-NOV-01            6
        15 29-JUN-10            6
        16 03-DEC-03            6
        17 15-JAN-85            6
        18 20-JUN-90            6
        19 20-FEB-10            6
        20 17-FEB-03            6

20 rows selected.
```

Logistics:

```
CS3420 SQL> desc eooo_logistics
Name              Null?    Type
--------------- -------- ----------
DEPARTMENTID      NOT NULL NUMBER(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_logistics;
DEPARTMENTID
------------
           2
```

Outreach:

```
CS3420 SQL> desc eooo_outreach
Name              Null?    Type
--------------- -------- ----------
DEPARTMENTID      NOT NULL NUMBER(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_outreach;
DEPARTMENTID
------------
           4
```

Parent:

```
CS3420 SQL> desc eooo_parent
Name                Null?    Type
--------------- -------- --------------
SSN                 NOT NULL NUMBER(9)
FNAME               NOT NULL VARCHAR2(30)
MNAME                   NULL VARCHAR2(30)
LNAME               NOT NULL VARCHAR2(30)
STREET              NOT NULL VARCHAR2(50)
CITY                NOT NULL VARCHAR2(30)
STATE               NOT NULL VARCHAR2(30)
ZIP                 NOT NULL NUMBER(5)
PHONE_NUMBER        NOT NULL NUMBER(10)
BDAY                NOT NULL DATE
PSEX                NOT NULL VARCHAR2(1)
STATUS              NOT NULL VARCHAR2(50)
INCOME              NOT NULL FLOAT(10)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_parent;
       SSN FNAME      MNAME   LNAME   ST       CITY     STAT        ZIP      PHONE BDAY      SEX  STAT     INCOME
---------- ---------- ------- ------- -------- -------- ---- ---------- ---------- --------- ---- ---- ----------
 111111111 Terry      Larry   Brooks  Hayes    Vancouve Wash      72822 6497925787 29-MAY-85 m    taxp  197000000
 222222222 Frank      Peter   Ruiz    Clyde Ga Omaha    Nebr      84573 7329191862 21-OCT-89 m    nont  726100000
 444444444 Shawn      Phillip Freeman Kenwood  Miami    Flor      56785 8199492304 14-MAR-79 m    taxp  286200000
 888888888 Tammy      Jean    Grant   Cambridg Arlingto Texa      76526 4788906278 09-OCT-73 f    nont  960900000
 111111113 Joshua     James   Fisher  Rockefel New York New       30829 2840443139 04-AUG-93 m    taxp    1506000
 222222224 Jose       Andrew  Hanson  Manufact Lafayett Indi      46133 6482193961 31-DEC-77 m    nont  721300000
 444444446 Sara       Andrea  James   Meadow V Worceste Mass      60352 9323045958 09-AUG-96 f    taxp  279800000
 888888890 Janet      Ashley  Bowman  Lakeland Pensacol Flor      54817 5172902674 18-OCT-90 f    nont  246100000
 111111115 Larry      Eugene  Ford    Oak      Montgome Alab      85404 4247743408 03-MAY-97 m    taxp  902600000
 222222226 Sara       Betty   Matthew Esch     Tallahas Flor      49183 4061676149 06-DEC-95 f    nont   59440000
 444444448 Melissa    Brenda  Bradley Arkansas Oklahoma Okla      63368 9738413302 27-MAY-89 f    taxp  953600000
 888888892 Elizabeth  Marie   Jones   Mcguire  San Fran Cali      30036 1888963106 27-JUL-96 f    nont  280400000
 111111117 Denise     Mary    Lewis   Stephen  Long Bea Cali      42002 8715068926 20-SEP-84 f    taxp  300800000
 222222228 Daniel     Justin  Roberts Southrid Saint Pe Flor      17950 6066168307 05-DEC-77 m    nont  577200000
 444444450 Sara       Amy     Pierce  Pleasure South Be Indi      31795 5645413759 28-SEP-97 f    taxp  979000000
 888888894 Alice      Gloria  Howell  Walton   San Bern Cali      71844 4786243430 17-AUG-75 f    nont  409600000
 111111119 Joseph     Samuel  Fisher  Merchant Las Vega Neva      92234 3993904243 14-DEC-82 m    taxp  725300000
 222222230 Susan      Anna    Lopez   Kropf    Houston  Texa      38295 5936080259 13-NOV-76 f    nont  593900000
 444444452 Wanda      Stephan Miller  Forest R Riversid Cali      91829 5976811965 21-APR-72 f    taxp  701800000
 888888896 Jeffrey    Russell Nelson  Hollow R Sacramen Cali      25365 6956517535 23-FEB-81 m    nont  536100000

20 rows selected.
```

Review_Board:

```
CS3420 SQL> desc eooo_reviewboard
Name              Null?    Type
--------------- -------- ---------
DEPARTMENTID      NOT NULL NUMBER(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_reviewboard;
DEPARTMENTID   SUBDEPID        BID
------------ ---------- ----------
           1          6          1
           2          6          2
           3          6          3
           4          6          4
           5          6          5
           6          6          6
           7          6          7
           8          6          8
           9          6          9
          10          6         10
          11          6         11
          12          6         12
          13          6         13
          14          6         14
          15          6         15
          16          6         16
          17          6         17
          18          6         18
          19          6         19
          20          6         20

20 rows selected.
```

School:

```
CS3420 SQL> desc eooo_school
Name               Null?     Type
--------------- -------- -------------
SCHOOL_ID          NOT NULL NUMBER(10)
NAME               NOT NULL VARCHAR2(50)
STREET             NOT NULL VARCHAR2(50)
CITY               NOT NULL VARCHAR2(30)
STATE              NOT NULL VARCHAR2(30)
ZIP                NOT NULL NUMBER(5)
DEPARTMENTID       NOT NULL NUMBER(1)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_school c where c.schoolid < 20;
SCHOOL_ID NAME     STREET   CITY     STATE                              ZIP DEPARTMENTID
--------- -------- -------- -------- ------------------------------ ---------- ------------
        1 Aaron    Northwes Tulsa    Oklahoma                           19509            4
        2 Clarence Manley   Santa Ba California                         62130            4
        3 Gregory  Loeprich Clearwat Florida                            82673            4
        4 Stephani Prairie  Cincinna Ohio                               62894            4
        5 Rebecca  Waxwing  Clevelan Ohio                               80327            4
        6 Anthony  Oakridge Orange   California                         87715            4
        7 Gloria   Harper   Phoenix  Arizona                            82564            4
        8 Michael  8th      Washingt District of Columbia               46465            4
        9 Lois     Westerfi Charlott North Carolina                     52638            4
       10 Steve    Sommers  Grand Ra Michigan                           37310            4
       11 Rose     Garrison Baltimor Maryland                           30231            4
       12 Ralph    Darwin   Providen Rhode Island                       59646            4
       13 Nancy    Paget    Loretto  Minnesota                          58791            4
       14 Diane    Upham    Charlott North Carolina                     21475            4
       15 Antonio  Ridgevie Bridgepo Connecticut                        54304            4
       16 Ruth     Acker    Reno     Nevada                             33848            4
       17 Kathleen Prentice Baltimor Maryland                           25293            4
       18 Diana    Di Loret Fresno   California                         35551            4
       19 Henry    Southrid Roanoke  Virginia                           68729            4
       20 Earl     Lukken   Philadel Pennsylvania                       33546            4

20 rows selected.
```

Student:

```
CS3420 SQL> desc eooo_student
Name                      Null?    Type
------------------------- -------- -------------
STUDENT_ID                NOT NULL NUMBER(10)
SSN                       NOT NULL NUMBER(9)
ACADEMIC_STANDING         NOT NULL VARCHAR2(1)
FNAME                     NOT NULL VARCHAR2(30)
MNAME                         NULL VARCHAR2(30)
LNAME                     NOT NULL VARCHAR2(30)
STREET                    NOT NULL VARCHAR2(50)
CITY                      NOT NULL VARCHAR2(30)
STATE                     NOT NULL VARCHAR2(30)
ZIP                       NOT NULL NUMBER(5)
PHONE_NUMBER              NOT NULL NUMBER(10)
SEX                       NOT NULL VARCHAR2(1)
INCOME_STATUS             NOT NULL VARCHAR2(50)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_student s where s.studentid < 20;
STUDENT_ID        SSN A FNAME      MNAME      LNAME   STREET   CITY       STAT      ZIP PHONE_NUMBER S INC
---------- ---------- - ---------- ---------- ------- -------- ---------- ---- -------- ------------ - ---
         1  125348809 g Robert     Karen      Cunning Red Clou Oakland    Cali    78781   8351582078 m dep
         2  909640508 b Scott      Kathleen   Bryant  La Folle Paterson   New     63481   8813081472 f ind
         3  651053529 g Catherine  Richard    Hawkins Bonner   Las Vegas  Neva    78251   2621881080 m dep
         4  248119719 b Teresa     Jacqueline William Clyde Ga Tucson     Ariz    80550   3664243333 f ind
         5  272994915 g Frances    Katherine  Cook    Farmco   Austin     Texa    85245   5057296799 m dep
         6  919264585 b Donald     Harold     Schmidt Fairfiel Kansas Cit Miss    55938   1881599866 f ind
         7  614956491 g Fred       Richard    Rodrigu Colorado Washington Dist    35518   1086793948 m dep
         8  762762036 b Brandon    Louise     Flores  Tony     Buffalo    New     56902   7397118631 f ind
         9  892102597 g Kenneth    Paul       Hunt    Truax    Richmond   Virg    23228   9279941433 m dep
        10  611956022 b Christina  Jerry      Davis   Melby    Whittier   Cali    81422   5017635650 f ind
        11  704413268 g Joseph     Frances    Morriso Talisman Columbus   Ohio    11974   4529590671 m dep
        12  851037973 b Fred       Angela     Howell  Pankratz Oklahoma C Okla    12515   9037051305 f ind
        13  317739602 g Tammy      Alan       Young   Crownhar Ocala      Flor    42251   4665995276 m dep
        14  614400292 b Philip     Frances    Black   Rusk     Aurora     Illi    11671   7602080345 f ind
        15  577486757 g Brandon    Cynthia    Hansen  Stone Co Washington Dist    41878   9921407770 m dep
        16  595694444 b Martha     Sandra     Frazier Washingt Charlottes Virg    61009   4205395189 f ind
        17  705826528 g Dennis     Angela     Jones   Debra    Sarasota   Flor    84000   7846443831 m dep
        18  199713655 b Martha     Betty      Lopez   Browning Columbia   Sout    34667   7490207913 f ind
        19  662406992 g Bruce      Gloria     Graham  Dixon    Lancaster  Cali    11660   1273754752 m dep
        20  237863006 b Gloria     Benjamin   Roberts Lindberg Washington Dist    31168   3952345886 f ind

20 rows selected.
```

Uses_Info_from:

```
CS3420 SQL> desc eooo_uinfof
Name                   Null?    Type
---------------------- -------- -------------
STUDENTID              NOT NULL NUMBER(10)
PARENTSSN              NOT NULL NUMBER(9)

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_uinfof u where u.studentid < 20;
STUDENTID  PARENTSSN
---------- ----------
         1  111111111
         2  222222222
         3  444444444
         4  888888888
         5  111111113
         6  222222224
         7  444444446
         8  888888890
         9  111111115
        10  222222226
        11  444444448
        12  888888892
        13  111111117
        14  222222228
        15  444444450
        16  888888894
        17  111111119
        18  222222230
        19  444444452
        20  888888896

20 rows selected.
```

Uses_Stores_Into:

```
CS3420 SQL> desc eooo_ustoresi
Name              Null?    Type
---------------- -------- --------------
DATAID            NOT NULL NUMBER(9)
DADESCRIPTION     NOT NULL VARCHAR2(30)
SOURCEID          NOT NULL NUMBER(9)
SOURCEDATE        NOT NULL DATE

CS3420 SQL> spool off
```

```
CS3420 SQL> select * from eooo_ustoresi u where u.dataid < 20;
    DATAID DADESCRIPTION                    SOURCEID SOURCEDAT
---------- ------------------------------ ---------- ---------
         1 parent                                  1 25-SEP-90
         2 student                                 2 30-AUG-09
         3 parent                                  3 15-JUL-99
         4 student                                 4 24-NOV-91
         5 parent                                  5 02-MAR-10
         6 student                                 6 16-DEC-03
         7 parent                                  7 05-JUN-10
         8 student                                 8 17-OCT-12
         9 parent                                  9 22-OCT-07
        10 student                                10 03-JUL-09
        11 parent                                 11 16-JUL-06
        12 student                                12 10-MAY-95
        13 parent                                 13 11-NOV-96
        14 student                                14 20-APR-98
        15 parent                                 15 11-JAN-98
        16 student                                16 10-APR-90
        17 parent                                 17 22-AUG-04
        18 student                                18 14-NOV-95
        19 parent                                 19 10-AUG-92
        20 student                                20 16-MAR-12

20 rows selected.
```

3.5 EXAMPLE QUERIES IN SQL

1. List students who received a Financial Aid Package of under 1,000 whose parents are taxpayers.

```
select s.student_id, s.fName from E000_STUDENT s
where exists (select d.studentID, d.faPackageID
from E000_DISTRIBUTEDTO d
where exists
(select fa.packageID, fa.amount from E000_FAPACKAGE fa
where exists (select * from E000_UINFOF ui natural join
E000_PARENT p
where (fa.packageID = d.faPackageID and fa.amount < 1000 and
s.student_id = d.studentID and ui.studentID = s.student_id
and ui.parentSsn = p.ssn ) ) ) )/

CS3420 SQL> @query1

STUDENT_ID FNAME
---------- ------------------------------
         8 Brandon
         9 Kenneth
        12 Fred
        13 John
        18 Martha


CS3420 SQL> spool off
```

2. List financial aid packages which contained the same amount for every student.

```
select * from EOOO_FAPACKAGE f, EOOO_DISTRIBUTEDTO d
where f.packageID = d.faPackageID and exists (select * from
EOOO_STUDENT s
where s.student_id = d.studentID and
exists (select * from EOOO_FAPACKAGE f2
where f2.packageID != d.faPackageID and f2.amount = f.amount));

CS3420 SQL> @query2

PID PTYPE      AMOUNT FID STUDENTID  FAPACKAGEID DDATE
--- ---------- ------ --- ---------- ---------- -----------
6   other      1848.2 6   6           6          10-JUL-00
7   calgranta  1848.2 7   7           7          27-MAY-96
10  calgranta  2426   10  10          10         19-SEP-96
14  calgrantb  2426   14  14          14         01-DEC-01
15  calgranta  1000   15  15          15         11-MAY-09
16  calgranta  1000   16  16          16         02-OCT-97
6 rows selected.


CS3420 SQL> spool off
```

3. List schools that were visited by Employees with Birthdays before 1992

```
select * from EOOO_SCHOOL s, EOOO_EMPLOYEE e, EOOO_DEPARTMENT d
where e.hire_date < to_date('01/01/1992', 'mm-dd-yyyy') and
s.departmentID = e.departmentID and e.departmentID =
d.departmentID and d.Name = 'Outreach';

CS3420 SQL> @query3

 SCHOOL_ID                NAME   STREET     CITY
---------- -------------------   ------ --------
       2    PabloEsco College      oak   Santa B
       2    PabloEsco College      oak   Santa B
       5    PabloEsco College   Waxwing  Clevela
       8    Michael                 8th  Washing
       11   Lois University    Garrison  Baltimo



CS3420 SQL> spool off
```

4. Find the most expensive budget in the history of the organization

```
select * from EOOO_BUDGET b
where not exists (select * from EOOO_BUDGET b2
where b2.amount > b.amount);

CS3420 SQL> @query4

  BUDGETID     AMOUNT BUDATE
---------- ---------- ---------
        19     999090 05-JAN-88

CS3420 SQL> spool off
```

5. Find the second most expensive budget in the history of the organization

```
select * from EOOO_BUDGET
where amount = (select max(amount) from EOOO_BUDGET
where amount < (select max(amount) from EOOO_BUDGET));

CS3420 SQL> @query5

  BUDGETID     AMOUNT BUDATE
---------- ---------- ---------
        18     989340 30-AUG-05

CS3420 SQL> spool off
```

6. List the academic standing of ALL students that did NOT receive Financial Aid (All the BAD STUDENTS).

```
select s.academic_standing from EOOO_STUDENT s
where exists(select * from EOOO_FILLSOUT f
where exists(select * from EOOO_APPLICATION a
where f.studentID = s.student_id
AND f.applicationID = a.applicationID AND a.approved = 'f'));

CS3420 SQL> @query6

academic_standing
-----------------
b
b
b
g
g
b
b
b
g

9 rows selected.
```

7. Find the schools which have ALL their student's Applications approved.

```
select * from eooo_school c, eooo_fillsout f, eooo_student s,
eooo_attending a, eooo_application p
where s.student_id = a.studentID and c.school_id = a.schoolID
and f.studentID = s.student_id and f.applicationID =
p.applicationID and p.approved = 't';

CS3420 SQL> @query7

 SCHOOL_ID NAME                  STREET        CITY
 --------- --------------------  ------------- -----
         1 PabloEsco College     Northwestern  Tulsa
         2 PabloEsco College     oak           Santa B
         3 PabloEsco College     Loeprich      Clearwa
         5 PabloEsco College     Waxwing       Clevela
         7 Gloria                Harper        Phoenix
         9 Lois University       Westerfield   Charlot
        11 Lois University       Garrison      Baltimo
        13 Nancy                 Paget         Loretto
        15 Antonio               Ridgeview     Bridgep
        17 Kathleen              Prentice      Baltimo
        19 Henry                 Southridge    Roanoke

11 rows selected.

CS3420 SQL> spool off
```

8. Find the school in which the student gets the Highest Financial Aid Award.

```
SELECT school_id, name
FROM EOOO_SCHOOL s NATURAL JOIN EOOO_ATTENDING a
WHERE s.school_id = a.schoolID
AND a.studentID = (SELECT studentID
  from EOOO_STUDENT s,EOOO_DISTRIBUTEDTO d
  WHERE s.student_id = d.studentID
  AND d.faPackageID = (SELECT packageID FROM EOOO_FAPACKAGE
  WHERE amount = (SELECT MAX(amount) FROM EOOO_FAPACKAGE)));

CS3420 SQL> @query8

 SCHOOL_ID NAME
---------- --------------------
         5 PabloEsco College

CS3420 SQL> spool off
```

9. Find the school in which ALL the students are in good academic standing (ALL GOOD STUDENTS)

```
select * from EOOO_SCHOOL c, EOOO_STUDENT s, EOOO_ATTENDING a
where s.academic_standing = 'g' and a.schoolID = c.school_id
and a.studentID = s.student_id;
CS3420 SQL> @query9

 SCHOOL_ID NAME                 STREET         CITY
---------- -------------------- -------------- -------
         1 PabloEsco College    Northwestern   Tulsa
         3 PabloEsco College    Loeprich       Clearwa
         5 PabloEsco College    Waxwing        Clevela
         7 Gloria               Harper         Phoenix
         9 Lois University      Westerfield    Charlot
        10 Lois University      Sommers        Grand R
        11 Lois University      Garrison       Baltimo
        12 Lois University      Darwin         Provide
        13 Nancy                Paget          Loretto
        15 Antonio              Ridgeview      Bridgep
        17 Kathleen             Prentice       Baltimo
        19 Henry                Southridge     Roanoke
        20 Earl                 Lukken         Philade

13 rows selected.

CS3420 SQL> spool off
```

10. List students who filled out an Application on each of ALL the days as students named John Doe did.

```
SELECT * FROM EOOO_STUDENT S, EOOO_FILLSOUT O
WHERE S.STUDENT_ID = O.STUDENTID
AND EXISTS (SELECT * FROM EOOO_STUDENT S2, EOOO_FILLSOUT O2
WHERE S2.STUDENT_ID = O2.STUDENTID AND O2.FDATE = O.FDATE
AND S2.FNAME = 'JOHN' AND S2.LNAME = 'DOE');

CS3420 SQL> @QUERY10


STUDENT_ID        SSN A FNAME      MNAME         LNAME   STR

---------- ---------- - ---------- ------------- ------- ----

        10  611956022 G CHRISTINA  JERRY         DAVIS   MEL

        12  851037973 G FRED       ANGELA        HOWELL  PAN

        13  317739602 G JOHN       ALAN          DOE     CRO


CS3420 SQL> SPOOL OFF
```

ADDITIONAL QUERIES (USING SQL*PLUS AND AGGREGATE FUNCTIONS)

The following Queries were added to demonstrate some SQL*PLUS features, such as ORDER BY. When using ORDER BY, it must be accompanied by an aggregate function, such as COUNT. Below are three (3) new queries showing these features.

11. List the number of students which attend PabloEsco College

```
select count(*) from eooo_student s, eooo_attending a,
eooo_school c
where s.student_id = a.studentID and a.schoolID = c.school_id and
c.name = 'PabloEsco Elementary'
order by count(*);

CS3420 SQL> @query11

  COUNT(*)
----------
         5

CS3420 SQL> spool off
```

12. Select the number of female students attending schools.

```
select count(*)
from eooo_school s, eooo_attending a, eooo_student e
where a.schoolID = s.school_id
and a.studentID = e.student_id and e.sex = 'f'
order by count(*);

CS3420 SQL> @query12

  COUNT(*)
----------
        10

CS3420 SQL> spool off
```

13. List the Parent's Names and Incomes whose income is less than the average income of all the parents.

```
select fName,income from eooo_parent
where income < (select avg(income) from eooo_parent)
order by income;

CS3420 SQL> @query13
FNAME                              INCOME
------------------------------ ----------
Joshua                            1506000
Sara                             59440000
Terry                           197000000
Janet                           246100000
Sara                            279800000
Elizabeth                       280400000
Shawn                           286200000
Denise                          300800000
Alice                           409600000
9 rows selected.
CS3420 SQL> spool off
```

3.6 Data Loader

There are a variety of ways to load large amounts of data into a physical implementation of the Database. Manually writing SQL commands and using software applications that create insert scripts from data are very common methods.

SQL STATEMENTS: "Insert"
The simplest way to insert Data into Oracle DBMS Tables is with the "insert" SQL statement.

Example:

```
1. INSERT INTO [table name]
         [column name 1 …. column name n]
   VALUES
         [expression 1 … expression n]

2. INSERT INTO [table name]
       [select query]
```

Number 1 lets one specify value expressions for each column in the Table when inserting a Record.
Number 2 lets one use the result of a Query as the Column values.

This method is not the best when loading large amounts of sample data into the Database. There exist alternatives for loading large amounts of Data faster.

Data Loader:

The Data Loader is Dr. Wang's software application which uses a command-line interface to insert data into Tables from a text file. The user of the application must specify the name of the Database, the Password, and the Text File to be used in the command line. The text file must follow a specific format, which specifies the data and the table into which the Tuple should be inserted. The user can also specify which character is used as a delimiter to separate columns through the command line (",", or "|"). "Insert into" SQL statements are generated and ran based on information in the text file.

Oracle SQL Developer:

Oracle SQL Developer is a software application that is provided by Oracle for free. SQL Developer allows users to develop and manage an Oracle Database. This is all achieved with a Graphical User Interface (GUI) that shows all the user's Tables. The SQL Developer GUI also has the feature of importing data into Tables with CSV files. This then returns an insert script with SQL commands if the user wants to paste the commands onto files. *Oracle SQL Developer* is what we used to load data in our project.

[Page Intentionally Left Blank]

Phase Four:
Oracle DBMS PL/SQL Components

The previous phase of our report demonstrated how a physical database is constructed in Oracle's Database Management System. This involved inserting mock data into the CS3420 QLPLUS Tablespace. Additionally, our queries were demonstrated in the SQL language to show how operations can be performed on the Data in our Database.

To follow Integrity Constraints and Business Rules, operations that are more complex are required. This phase explores the implementation of complex operations in Oracle's procedural extension of the SQL Language: *PL/SQL.*

First we will explain the purpose of *PL/SQL* and the Syntax for utilizing each. Second, we will introduce *PL/SQL* features and the syntax for them. Third, we will implement *PL/SQL* operations for our Financial Aid Database. Finally, we will introduce and describe extensions of *SQL* offered by other Database Management Systems and compare them with *PL/SQL*.

4.1 Oracle PL/SQL

PL/SQL is a procedural language extension of SQL created by Oracle. Users implementing *PL/SQL* features can define the order in which SQL statements are executed with the use of conditional statements and loops (*flow control structures*).

PL/SQL is used to build *Stored Procedures* and *Functions*, which are precompiled blocks of *PL/SQL* code that can be run at any time.

The use of *Stored Procedures* in a Database application has many advantages over writing *PL/SQL* blocks manually and sending them to the server:
1. Stored Procedures are *precompiled*, meaning that the *PL/SQL* code does not need to be compiled every time it is ran. Precompiling saves time during execution.
2. Stored Procedures are *reusable*, meaning that they operate like *functions* which can be repeatedly used by different users.
3. Stored Procedures hide a lot of complexity from users, resulting in simpler and safer code-writing.

4.1.1 Program Structure and Control Statements

Oracle PL/SQL is a block structured language in which the functions, procedures and anonymous blocks are the basic blocks.
Oracle PL/SQL also supports the three types of control structures: *sequence*, *selection*, and *iteration.*

Syntax for anonymous blocks:

```
DECLARE
     [variables]
BEGIN
     [PL/SQL statements]
END
```

Oracle PL/SQL's control structures consist of *conditional statements*, and different types of *loops*.

Conditional Statement syntax:

```
IF <condition> THEN
     [statements]
END IF
IF <condition> THEN
     [statements]
ELSEIF <condition> THEN
     [statements]
END IF;
EXIT-WHEN condition;
```

Loop syntax:

```
LOOP
        [statements]
END LOOP;
FOR i IN lowerbound … upperbound LOOP
         [statements]
END LOOP;

FOR cursos_variable in cursor_name LOOP
         [statements]
END LOOP;

WHILE condition LOOP
         [statements]
END LOOP;
```

4.1.2 Stored Procedures

Stored Procedures are like procedures in other programming languages. They contain a PL/SQL block which performs one or more specific tasks. Procedures contain a header, which consists of the name of the procedure and the parameters/variables passed to it, and a body, which consists of the declaration, execution, and exception sections. Stored procedures differ from stored functions (see below) because they do not return a value.

Syntax of a Stored Procedure:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
   [list of parameters]
AS
   Declaration section
BEGIN
   Execution section
EXCEPTION
   Exception section
END;
```

4.1.3 Stored Functions

Stored Functions are the same as Stored Procedures, except that Stored Functions return values. This is like functions in other programming languages that return a value based on the data type of the function.

Syntax of a Stored Function:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
   [list of parameters]
RETURN [return type]
AS
   Declaration section
BEGIN
   Execution section
EXCEPTION
   Exception section
END;
```

4.1.4 Packages

A package is a database object that groups related type definitions, objects and subprograms together. Packages have the advantage of modularity, easier application design, information hiding, added functionality, and better performance.
Modularity: Related functionality can be gathered and stored together just like C library functions.

Easier Application Design: Package specification can be created without the implementation of the package body.

Information Hiding: Hiding of subprogram implementation, private data types, and cursors.

Added functionality: Packaged public variables and cursors persists for the duration of a session. These can be shared among all subprograms. Global variables allow one to maintain data across transactions without having to store it in the Database.

Better Performance: Once a packaged subprogram is called for the first time, it may remain in Memory for some time. Subsequent calls to that subprogram or related subprograms will probably not require any I/O.

A package consists of two parts:
1. **Package specification** – The package specification part declares the constants, variables, types, exceptions, cursors and subprogram that will be callable from outside the package.
2. **Body** - The body of the package is the implementation of a package. This is where cursors and subprograms are defined.

Syntax of a Package:

```
CREATE [OR REPLACE] PACKAGE package_name
AS
    [function, procedure, object prototypes]
END;

CREATE [OR REPLACE] PACKAGE BODY package_name
AS
   [function, procedure, object definitions]
END;
```

4.1.5 Triggers

A trigger is a stored PL/SQL procedure that has an association with the Database, a Schema, a Table, or a View. Triggers automatically execute before, after, or instead of an event. Cascade Deletion requires the use of triggers. When a Tuple has a Key that is referenced by other Tuples, the other Tuples must first be deleted.

Syntax of a Trigger:

```
CREATE [OR REPLACE] TRIGGER trigger_name
[before, after, instead of] [event_name] ON [table_name]
FOR EACH ROW
BEGIN
        [PL/SQL statements]
END
```

4.2 Oracle PL/SQL Subprogram Examples

In this section, we will implement a package, as well as procedures/functions and triggers on our Database. The purpose of a package is to group all the procedures and functions into one unit. Our procedures will *insert* a student, *delete* a student, and *calculate the average* income for all parents. Additionally, three triggers will be implemented demonstrating the *cascade delete*, *update*, and *instead of* trigger to update a *view*.

Package:

A package serves the purpose of grouping functions, procedures, type definitions, and other Oracle objects into one unit. Members of a package are in a separate namespace than the rest of the objects in the Database. This leads to conflict avoidance.

Packages contain a *header* that includes prototypes for all the procedures and functions. Packages also contain a *body* which defines and implements all the procedures and functions.

Our example package, *eooo_pkg* is shown below. Our package contains procedures *delete_student, insert_student*, and *average_income* function. The details for each are listed below.

```
CS3420 SQL> @eooo_pkg_header
Package created.

CS3420 SQL> list
create or replace package eooo_pkg as

procedure delete_student (
    sid in eooo_student.student_id%type
);

procedure insert_student (
    ssn in eooo_student.ssn%type,
    fn in eooo_student.fName%type,
    mn in eooo_student.mName%type,
    ln in eooo_student.lName%type,
    st in eooo_student.street%type,
    c in eooo_student.city%type,
    s in eooo_student.state%type,
    z in eooo_student.zip%type,
    p in eooo_student.phone_number%type,
    x in eooo_student.sex%type,
    ic in eooo_student.income_status%type
);

function average_income
(
    n number default 1
)
return number;

end eooo_pkg;
CS3420 SQL> spool off
```

Insert Student Procedure Definition:

The *insert_student* procedure inserts a Student record into the Database. When this procedure executes, all arguments passed into the call will be the attributes for *eooo_student*. *Student_id* attribute is not passed in as an argument because the procedure will find the *max* value of *student_id* and increment it by one. Once the *student_id* has been incremented, all the new attribute values will be inserted into that table's row.

```
procedure insert_student (
 ssn in eooo_student.ssn%type,
 acs in eooo_student.academic_standing%type,
 fn in eooo_student.fName%type,
 mn in eooo_student.mName%type,
 ln in eooo_student.lName%type,
 st in eooo_student.street%type,
 c in eooo_student.city%type,
 s in eooo_student.state%type,
 z in eooo_student.zip%type,
 p in eooo_student.phone_number%type,
 x in eooo_student.sex%type,
 ic in eooo_student.income_status%type)
is
   next_id eooo_student.student_id%type;
begin
    select max(s.student_id) into next_id from eooo_student s;
    next_id := next_id + 1;
    insert into eooo_student(
        student_id,
        ssn,
        academic_standing,
        fName,
        mName,
        lName,
        street,
        city,
        state,
        zip,
        phone_number,
        sex,
        income_status
    )values(next_id, ssn, trim(acs), trim(fn), trim(mn), trim(ln),    trim(st),
trim(c),trim(s), trim(z), trim(p), trim(x), trim(ic));
      commit;
exception
        when others then
            rollback;
            dbms_output.put_line(sqlcode || ', ' || sqlerrm);
         commit;
end insert_student;
```

Insert Student Procedure execution and results:

```
CS3420 SQL> exec eooo_pkg.insert_student(987654321, 'g',
'Omar','Oseg','5313ONE','Bakersf','CA',94412,6613124567,'m','ind');


PL/SQL procedure successfully completed.


CS3420 SQL> select * from eooo_student where fName = 'Omar';


SID SSN     A F    L    STREET  CITY    STATE ZIP  PHONE    S ISTAT

--- ------- - ---- ---- ------- ----    ----- ---- -------  - -----

21  98765   g Omar Oseg 5313ONE Bakersf CA    9441 3124567  m   ind


CS3420 SQL> spool off
```

Delete Student Procedure Definition:

eooo_delete_student procedure deletes a Tuple of *eooo_student* upon execution. This procedure takes a parameter which will be the *student_id* identifying the Tuple to delete. For this procedure to work properly, all the tables that use *student_id* as a Foreign Key must first be deleted.

```
CS3420 SQL> list
create or replace procedure delete_student(
    sid in eooo_student.student_id%type
)
is
begin
delete from eooo_student i
where i.student_id = sid;
     commit;
end delete_student;
CS3420 SQL> spool off
```

Before Delete Student Trigger Definition:

For the *eooo_delete_student* procedure to work properly, any table including *student_id* as a Foreign Key must also be deleted. The *before_delete_student* trigger runs automatically when the *eooo_delete_student* is called, and it will delete all Tuples associated with *student_id* via a Foreign Key before deleting the actual *eooo_student* Tuple.

```
CS3420 SQL> @before_delete_student
Trigger created.

CS3420 SQL> list

create or replace trigger eooo_delete_student

before delete on eooo_student

for each row
begin
    delete from eooo_attending at
    where at.studentID = :old.student_id;

    delete from eooo_distributedto dt
    where dt.studentID = :old.student_id;

    delete from eooo_fillsout fo
    where fo.studentID = :old.student_id;

    delete from eooo_uinfof nf
    where nf.studentID = :old.student_id;

    exception
        when others then
        rollback;
        dbms_output.put_line(sqlcode || ', ' || sqlerrm);
        commit;
end;

CS3420 SQL> spool off
```

Delete Procedure Execution and Results:

Once the *eooo_delete_student* is executed with an argument, the Tuple (whose *student_id* corresponds to the value passed in for the argument) will be deleted. Additionally, all Tuples associated with the *student_id* will also be deleted. This trigger then acts as a *cascade delete* throughout the database.

```
CS3420 SQL> exec eooo_pkg.delete_student(7);
PL/SQL procedure successfully completed.
CS3420 SQL> select student_id, fName, lName from eooo_student;
STUDENT_ID FNAME                          LNAME
---------- ------------------------------ ---------------------
         1 Robert                         Cunningham
         2 Scott                          Bryant
         3 Catherine                      Hawkins
         4 Teresa                         Williamson
         6 Donald                         Schmidt
         8 Brandon                        Flores
         9 Kenneth                        Hunt
        10 Christina                      Davis
        11 Joseph                         Morrison
        12 Fred                           Howell
        13 John                           Doe
        14 Philip                         Black
        15 Brandon                        Hansen
        16 Martha                         Frazier
        17 Dennis                         Jones
        18 Martha                         Lopez
        19 Bruce                          Graham
        20 Gloria                         Roberts
18 rows selected.
CS3420 SQL> spool off
```

Calculate Average Income Function:

The *average_income* function returns the average income from a certain number of Parent Tuples. The number of Tuples is determined by a value passed through the parameter *n*. *Order By* and *rownum* are used to retrieve only the top *n* Tuples. The aggregate function *average* is used to find the average from the *income* values.

```
CS3420 SQL> select eooo_pkg.average_income(10) from dual;
EOOO_PKG.AVERAGE_INCOME(10)
---------------------------
               784170000
CS3420 SQL> spool off
```

Instead of Trigger Definition:

The *insteadof* trigger is used to control update operations on views that join two or more tables. The *insteadof* trigger ensures that the base tables are updated instead of the view when an update operation is executed. Our example will use the *insteadof* trigger to handle updates on a view we created joining the *eooo_student* and *eooo_distributedto* tables. With this trigger, either *eooo_student* or *eooo_distributedto* are updated (or new Tuple is created) based on the value of the *student_id*.

```
CS3420 SQL> @insteadof
Trigger created.
CS3420 SQL> list
create or replace trigger eooo_stu_distri_inf_update
instead of update on eooo_student_distribution_info
for each row
declare
    cnt number;
begin
    /*see if student id references an existing tuple*/
    select count(*) into cnt from eooo_student
    where student_id = :new.student_id;

    if cnt = 0 then
        /*tuple does not exist, create new tuple*/
        insert into eooo_student (student_id, fname, lName)
        values(:new.student_id, :new.fName, :new.lName);
    else
        /*if tuple exists, then update it*/
        update eooo_student st set st.fName = :new.fName,
        st.lName = :new.lName
        where st.student_id = :new.student_id;
    end if;

  /*update distribution with attributes from distribution table */
    update eooo_distributedto d
    set d.studentID = :new.student_id, d.dDate = :new.dDate
    where d.faPackageID = :old.faPackageID;

exception
when others then
    rollback;
    dbms_output.put_line(sqlcode || ', ' || sqlerrm);
    commit;
end;

CS3420 SQL> spool off
```

Instead of Trigger Execution and Results:

In this example, an update operation was performed on *eooo_student_distribution_info*, and the *eooo_student* and *eooo_distributedto* attributes received updates.

```
CS3420 SQL> update eooo_student_distribution_info set student_id = 4,
            fName = 'Erik', lName = 'Ort' where faPackageID = 4;
1 row updated.
CS3420 SQL> select student_id, fName, lName from eooo_student where
student_id = 4;
STUDENT_ID  FNAME  LNAME
----------  -----  ------
         4  Erik   Ort
CS3420 SQL> select * from eooo_student where student_id = 4;

STUDENT_ID        SSN A FNAME MNAME      LNAME  STR
---------- ---------- - ----- ---------- -----  -----
         4  248119719 b Erik  Jacqueline  Ort   Cly


CS3420 SQL> select * from eooo_distributedto where faPackageID = 4;
 STUDENTID FAPACKAGEID DDATE
---------- ----------- ---------
         4           4 07-MAR-02
CS3420 SQL> spool off
```

Update Trigger Definition:

The update Trigger is used to ensure that once a Primary Key of a Tuple is changed in *eooo_student*, then that value is also changed for the Foreign Key attributes of any Tuple that references *eooo_student*.

```
CS3420 SQL> @before_update_student

Trigger created.

CS3420 SQL> list
create or replace trigger eooo_update_student
before update on eooo_student
for each row
begin
    update eooo_attending at
    set at.studentID = :new.student_id
    where at.studentID = :old.student_id;

    update eooo_distributedto dt
    set dt.studentID = :new.student_id
    where dt.studentID = :old.student_id;

    update eooo_fillsout fo
    set fo.studentID = :new.student_id
    where fo.studentID = :old.student_id;

    update eooo_uinfof nf
    set nf.studentID = :new.student_id
    where nf.studentID = :old.student_id;

exception
    when others then
        rollback;
        dbms_output.put_line(sqlcode || ', ' || sqlerrm);
        commit;
end;

CS3420 SQL> spool off
```

Update Trigger Execution and Results:
In this example, a *student_id* value of *88* was changed to *22*. This sets off the update trigger which changes the *student_id* of all Tuples referencing the value *88* from other tables.

```
CS3420 SQL> update eooo_student set student_id = 22
            where student_id = 88;

1 row updated.

CS3420 SQL> select student_id as sid, fName, lName from eooo_student
            where student_id = 22;

  SID  FNAME          LNAME
-----  -----------    -----------------
   22  Brandon        Flores

CS3420 SQL> select student_id as sid, fName, lName from eooo_student
            where student_id = 88;

no rows selected
CS3420 SQL> spool off
```

4.3 PL/SQL Like Tools (Oracle, Microsoft SQL Server, MySQL)

Oracle PL/SQL was utilized to implement our physical database and procedures on that database. There is other commercial DBMS software that offers various functionalities. In this section Oracle PL/SQL, Microsoft SQL Server Transact-SQL, and MySQL will compared to one another regarding stored procedure functionality and syntax.

Microsoft SQL Server: T-SQL

Comparing other DBMS Languages:
T-SQL for Microsoft SQL Server offers unique functionality in comparison to other DBMS languages. In this DBMS, there are various options which enable restricting user permissions and other options which enable the encryption of the text of the procedure. T-SQL allows for multiple *try-catch* blocks in a procedure. T-SQL can return tables and scalar values without any complications. Of course, Oracle has this same functionality, however it is much more difficult to implement.
To pass and use parameters in T-SQL, the character, '@' must precede all parameters. This contrasts MySQL and PL/SQL.
While T-SQL provides many advantages in comparison to other DBMS languages, it also lacks some functionality. For example, T-SQL does not have the functionality of implementing for loops. Only basic loops and while loops are available for implementation. Additionally, T-SQL does not allow procedures to be grouped into packages.

Syntax for T-SQL Procedure: from msdn.microsoft.com developer website

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
   [ { @parameter [ type_schema_name. ] data_type }
   [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
   ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]

<procedure_option> ::=
   [ ENCRYPTION ]
   [ RECOMPILE ]
   [ EXECUTE AS Clause ]
```

Syntax for T-SQL Scalar Function: from msdn.microsoft.com developer website

```
CREATE FUNCTION [ schema_name. ] function_name

([ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
   [ = default ] [ READONLY ] }
   [ ,...n ]
 ]
)
RETURNS return_data_type
 [ WITH <function_option> [ ,...n ] ]
 [ AS ]
 BEGIN
   function_body
   RETURN scalar_expression
 END
[ ; ]
```

Syntax for T-SQL loop: from msdn.microsoft.com developer website

```
WHILE Boolean_expression
 { sql_statement | statement_block | BREAK | CONTINUE }
```

MySQL

Comparison other DBMS languages:

Offering similar functionality to PL/SQL and T-SQL, MySQL offers most of the essential control structures but does not offer for loops. Only while loops and basic loops are available for implementation. Unlike PL/SQL, MySQL does not offer packages for namespace management.

Parameters are passed similarly to PL/SQL. When creating a procedure in MySQL, the delimiter command must be used to change the default end-line character from a semicolon ‘;’ to “//”. Otherwise, only the initial line of the procedure will be stored.

Syntax for MySQL Procedure: from dev.mysql.com developer website

```
CREATE
      [DEFINER = { user | CURRENT_USER }]
      PROCEDURE sp_name ([proc_parameter[,...]])
      [characteristic ...] routine_body

CREATE
      [DEFINER = { user | CURRENT_USER }]
      FUNCTION sp_name ([func_parameter[,...]])
      RETURNS type
      [characteristic ...] routine_body
```

Syntax for MySQL loop: from dev.mysql.com developer website

```
[begin_label:] LOOP
      statement_list
END LOOP [end_label]
```

Oracle: PL/SQL

Comparison to other commercial DBMS languages:

Oracle's procedural SQL-based language for Oracle DBMS, PL/SQL, implemented the physical database for California Aid. PL/SQL provides several exclusive features compared to the other DBMS languages. Some of these features include packages, which prevents name conflicts. The parameter-passing mechanism is very similar to the structure of MySQL.

Syntax for PL/SQL creating a procedure/function:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
   [list of parameters]
AS
   Declaration section
BEGIN
   Execution section
EXCEPTION
   Exception section
END;
```

Syntax for PL/SQL loops:

```
LOOP
        [statements]
END LOOP;
FOR i IN lowerbound … upperbound LOOP
         [statements]
END LOOP;

FOR cursos_variable in cursor_name LOOP
         [statements]
END LOOP;

WHILE condition LOOP
         [statements]
END LOOP;
```