

California State University, San Bernardino

**CSUSB ScholarWorks**

---

Theses Digitization Project

John M. Pfau Library

---

2000

## Billing and receivables database application

Sushma Lukalapu

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Data Storage Systems Commons](#)

---

### Recommended Citation

Lukalapu, Sushma, "Billing and receivables database application" (2000). *Theses Digitization Project*. 1618.

<https://scholarworks.lib.csusb.edu/etd-project/1618>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact [scholarworks@csusb.edu](mailto:scholarworks@csusb.edu).

BILLING AND RECEIVABLES

DATABASE APPLICATION

---

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Sushma Lukalapu

June 2000

BILLING AND RECEIVABLES

DATABASE APPLICATION

---

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Sushma Lukalapu


June 2000

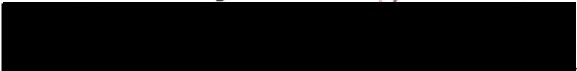
Approved by

  
Dr. Josephine Mendoza, Chair

3-15-2000  
Date

  
Dr. George M. Georgiou

  
Dr. Tong Lai Yu

  
Ms. Lorraine Frost

## ABSTRACT

The Accounting Department at California State University, San Bernardino (CSUSB), keeps track of vast amount of student financial data. The purpose of the Billing/Receivables application is to develop a computerized database system to keep track of all the financial details pertaining to students such as tuition, registration, parking, housing, etc. Originally, all the data has been stored in files. This application has been converted into a database from a file-based system. The application aims to support a centralized approach to the billing and receivables functions, process and track the transactions, and also provide ad-hoc access to produce appropriate reports. This database offers several benefits over the current file based system: allows multiple user access; reduces data redundancy; provides speedy recovery of data; maintains data integrity; allows security restrictions; and balances conflicting requirements. The database is designed using the entity-relationship (E-R) model concepts. The database is implemented using a relational database management system - ORACLE 7.3. along with its components SQL\*Plus 3.1., SQL\*Forms 4.5., and SQL\*Reports 2.5. The database executes as an integrated menu



application (designed using SQL\*Forms). It also includes user-friendly data entry forms, and several SQL queries that generate organized reports of use requested information. The database has been validated for its functionality and is ready for use.

## ACKNOWLEDGMENT

I thank the faculty of the Computer Science department for giving me an opportunity to pursue my M.S. in Computer Science at California State University, San Bernardino. I express my sincere appreciation to my graduate advisor, Dr. Josephine Mendoza, who provided me with invaluable guidance through this entire effort. I also thank my other committee members, Dr. George M. Georgiou and Ms. Lorraine Frost, for their valuable input. I would also like to thank the staff of the Administrative Computing Services for their cooperation and advice in the analysis of the project. Last but not least, I would like to thank my husband Ramana for his invaluable support and my friend Air Radha for his help with Oracle applications.

## TABLE OF CONTENTS

1. INTRODUCTION .....	1
1.1.INTRODUCTION .....	1
1.2.PURPOSE OF THE PROJECT .....	2
1.3.RESULTS OF THE PROJECT .....	3
2. DATABASE REQUIREMENTS AND SPECIFICATIONS .....	5
2.1.PROJECT APPROACH .....	5
2.2.INFORMATION DESCRIPTION .....	6
2.2.1.Information Flow Representation.....	6
2.2.2.Flaws in the Existing File Based System:.....	14
2.2.3.Information Content Representation.....	15
2.2.4.System Interface Description.....	15
2.2.5.Conceptual Database Design.....	18
2.3.SYSTEM REFERENCE .....	42
2.4.SOFTWARE PROJECT CONSTRAINTS .....	42
2.5.FUNCTIONAL DESCRIPTION .....	43
2.5.1.Functional Description.....	43
2.6.BEHAVIORAL DESCRIPTION .....	46
2.6.1.System State.....	46
2.6.2.Events and Actions.....	46
2.7.VALIDATION CRITERIA .....	49
2.7.1.Performance Bounds.....	49
2.7.2.Classes of Tests.....	50
2.7.3.Expected Software Response.....	50
2.7.4.Special Considerations.....	50
3. PHYSICAL DATABASE DESIGN.....	51
3.1.CREATION OF TABLES .....	51
3.2.DATA LOADING .....	55

3.3.DESIGN of SQL*FORMS .....	56
3.4.DESIGN OF SQL*REPORTS .....	58
3.5.FUTURE ENHANCEMENTS .....	60
4. DATABASE VALIDATION.....	61
4.1.UNIT TESTING .....	61
4.2.INTEGRATION TESTING .....	62
4.3.SYSTEM TESTING .....	62
APPENDIX A: CREATING TABLES, FORMS AND REPORTS .....	63
APPENDIX B: STEPS TO GENERATE A MASTER DETAIL FORM .....	70
APPENDIX C: FORMS CREATED FOR PROJECT .....	75
APPENDIX D: REPORTS CREATED FOR PROJECT .....	94
BIBLIOGRAPHY .....	104

## LIST OF TABLES

2.2.4.2.1.1. ATTRIBUTE DETAILS FOR STUDENT ENTITY.....	24
2.2.4.2.1.2. ATTRIBUTE DETAILS FOR CHARGES ENTITY.....	26
2.2.4.2.1.3. ATTRIBUTE DETAILS FOR PAYMENT ENTITY.....	27
2.2.4.2.1.4. ATTRIBUTE DETAILS FOR ACCOUNT ENTITY.....	28
2.2.4.2.2.1. ATTRIBUTE DETAILS FOR BILLED RELATIONSHIP.....	31
2.2.4.2.2.2. ATTRIBUTE DETAILS FOR APPLIED RELATIONSHIP.....	34
2.6.2. PRIMARY AND FOREIGN KEYS.....	43

LIST OF FIGURES

FIGURE 2.2.3. INFORMATION CONTENT REPRESENTATION-EXAMPLE...	16
FIGURE 2.2.5.2. ENTITY RELATIONSHIP DIAGRAM.....	22

## **1. INTRODUCTION**

### **1.1. INTRODUCTION**

The Accounting Department at California State University, San Bernardino (CSUSB) maintains the financial details of the student accounts like tuition, boarding, housing, parking, etc. The department has to keep track of an extensive amount of student account information. This requirement clearly drives the need for a computerized database system that allows a greater level of flexibility in the organization and storage of data, maintenance of data, and retrieval of data. This need for a more versatile database led to the development of this project, the Billing and Receivables database application. The Billing/Receivables application aims to:

- Support a centralized approach to the billing and receivables functions
- Process and track Billing/Receivables transactions
- Facilitate decision making
- Facilitate follow-up
- Provide ad-hoc access to produce appropriate reports

## 1.2. PURPOSE OF THE PROJECT

The purpose of this project is to design, build, and implement an information retrieval database system for the Accounting Department at CSUSB. The database will focus on the financial details of the student accounts maintained by the accounting personnel. It offers detailed information pertinent to tuition, parking, housing, boarding, etc. The main reason for selecting the database approach over the existing file-based approach is due to specific advantages of centralized control of data. Some advantages of the database approach are:

- Redundancy can be controlled
- Data can be shared
- Consistency of data is maintained
- Multiple users can access the data at a time
- Standards can be enforced
- Security is improved
- Data integrity can be maintained
- Conflicting requirements can be balanced



- Backup and recovery services can be improved
- Concurrency can be increased

The database is implemented using a commercial database management system - ORACLE Version 7.3. This database will address the immediate needs of the accounting staff and at the same time would also permit the accounting personnel to further enhance the application.

### 1.3. RESULTS OF THE PROJECT

This project consists of the following outputs:

Database Application: A working database with relevant application programs, that meet the specific needs of the Accounting Department with respect to storage and retrieval of student accounts. ORACLE SQL\*Forms and SQL\*Reports are utilized to maximize the user-friendliness of the database application.

Users Manual: An implementation manual will be provided for the users.

Data Dictionary: A dictionary detailing all the fields will be provided.

Systems Manual: A project report containing the details

and specifications of the design will be available; and  
procedure to create template reports and forms will be  
provided.

## 2. DATABASE REQUIREMENTS AND SPECIFICATIONS

### 2.1. PROJECT APPROACH

The campus accounting department's B/R applications are currently functioning on an IBM-4391, mainframe system. This computer system adequately supports the administrative applications with the help of upgrades and enhancements. However, this IBM system doesn't adequately support the efficient retrieval of data from the administrative systems for ad-hoc reporting purposes. The application consists of various screens written in COBOL. The student billing information is viewed by means of screens and the data updated on the screens is directly updated in the VSAM (Virtual Sequential Access Method) files on the mainframe system. However, other information not found on the screens is obtained by sending the requests to the COBOL analysts in the Administrative computing. These analysts write COBOL programs to get the required data and send the reports back to the accounting department. This procedure is not very efficient because the accounting department personnel have to wait for a day before they get the required information. Motivation for this project comes from wanting to develop a better, faster and complete database for the needs of the

end users in the accounting department. The ad-hoc queries will be done by accounting personnel as required using BRIO. BRIO is a query language developed at BRIO Technology, Inc. to retrieve data. Accounting personnel are trained in using BRIO.

A review of different types of data used by the accounting staff revealed that the data constituted distinguishable objects (entities) that can be linked to each other with certain relationships. Therefore, the concept of relational model was chosen as the design approach. The relational model is a way of representing data in tables and manipulating the data by means of operators such as Select, Join, Insert, Delete, Update, etc.

## 2.2. INFORMATION DESCRIPTION

### 2.2.1. Information Flow Representation

The data for the database design has been obtained after a thorough analysis of the existing data files and discussions with the current users and analysts. All their requirements and specifications were taken into account while designing the database.

To start with, twelve files were obtained from the accounting department:

FILE NAME	FILE DESCRIPTION
AA	Student attribute file - contains the personal information of a student
AM	Student miscellaneous file - contains information like who did the transaction on a student, the holds if any on students, etc.
AP	Building/room posting file - contains information about the courses being offered and the location of each class, etc.
AR	Tuition rate table file - contains information of how much to charge a student depending on the number of units enrolled, flat fees, etc.
BC	Subcode file - contains information of the subcode for a specific type of charge, the subcode description, if the transaction is a charge or payment, etc. Every subcode is associated with a 10 digit account number

FILE NAME	FILE DESCRIPTION
BD	Financial detail file - contains information about a bill, e.g., the date bill was prepared, the amount, person who prepared the bill, due date of the bill, etc.
BM	Cash checkout file - if the payments are in cash, who took the payments and were they posted to the account, etc.
BS	Financial summary file - contains information about a bill, e.g., the bill type, if the transaction has been fed to the corresponding accounts, etc.
CF	Cashiering fees feed file - corresponds to the cashiers office
CS	Cashiering suspend/line item file - corresponds to the cashiers office

FILE NAME	FILE DESCRIPTION
RC	Course term file - contains information about a particular course like the course no., units, name, faculty offering the course, etc.
RT	Student term file - contains information of the courses taken by a student in a particular quarter, whether the student is a sophomore, a graduate, etc.

After thorough analysis of all the above files, it was concluded that AA, BD and BS are the only files related to the B/R module. After discussions with the end users, all the required attributes were selected from the files. The following is the detailed description of attributes in the AA, BD, and BS files.

FILE AA - STUDENT ATTRIBUTE FILE\

ATTRIBUTE NAME	ATTRIBUTE DEFINITION
SID	Social security number

ATTRIBUTE NAME	ATTRIBUTE DEFINITION
PrevSID	Previous social security number if any or blank
Name	Student name
PrevName	Previous student name if any or blank

FILE BD - FINANCIAL DETAIL FILE

ATTRIBUTE NAME	ATTRIBUTE DEFINITION
TranDate	Date transaction was done
SID	Social security number
ExtrInd	External indicator. Specifies whether the student is regular ('-'), contract ('C'), or extension ('E').
Subcode	subcode number assigned to the charge or payment



ATTRIBUTE NAME	ATTRIBUTE DEFINITION
Transeq	sequence number of the transaction
BillDate	date detail item was prepared
TranAcctRef	user defined id for the transaction
TranAmt	amount charged for a subcode
PaidDate	date the transaction was paid
PaidAmt	total amount paid
TranUser	code indicating the user of the transaction
EffDate	date when charge or payment is effective
AcctngFeed	whether transaction fed to accounting system  '-': not done; '2': first phase done; '3': check written for refund
DueDate	avoid penalty date

ATTRIBUTE NAME	ATTRIBUTE DEFINITION
ChrgClass	control the order in which payments are applied to charges during the accounting feed
TranAcctl	charge debited from this account
TranAcct2	charge credited to this account
ReceiptNum	number on the receipt of the transaction

FILE BS - SUBCODE DESCRIPTION FILE

ATTRIBUTE NAME	ATTRIBUTE DEFINITION
SubDesc	description of the subcode
Chrg_Pay_Ind	whether charge of payment
APFeedInd	INDICATES TRANSACTION WITH THIS SUBCODE TO ACCOUNTS PAYABLE (A/P); 'N': NO REFUND; 'Y' : REFUND, CHECKS WILL BE GENERATED FROM AN A/P VOUCHER

ATTRIBUTE NAME	ATTRIBUTE DEFINITION
BillType	<p>indicates how transactions posted to the subcode are to be processed</p> <p>`-`: nothing;</p> <p>`I`: tuition income clearing account;</p> <p>`T`: non-resident tuition;</p> <p>`N`: Other tuition;</p> <p>`Q`: other fees;</p> <p>`F`: registraton;</p> <p>`B`: boarding;</p> <p>`H`: housing;</p> <p>`C`: contract;</p> <p>`S`: scholarship;</p> <p>`E`: exemption;</p> <p>`W`: waiver;</p> <p>`D`: deposit;</p> <p>`A`: financial aid;</p> <p>`C`, `S` and `E` are used during tuition calculation to generate credit forms.</p>

### 2.2.2. Flaws in the Existing File Based System:

- Field Charge Class is 3 characters long which keeps track of the order in which the payments are applied. The data being entered is not very understandable. However, after another discussion with the analysts, Chrg Class is a three letter id which tells what payment has to be applied when. It is the order of the payments applied to the account. It can take values A, B, and AB. However, a look-up table is required that describes the meaning of A, B, and AB.
- Transaction User is 2 characters and it is the code indicating the user. There should be some kind of a look up table that specifies the user name corresponding to the code.
- Bill Date is redundant because most of the time it is the same as the Effective Date.
- Transaction account reference is the user defined id for the transaction. There should be some document to keep track of all the ids. Also there is no clear need for this id. As there is transaction sequence which keeps

track of the transaction.

#### 2.2.3. Information Content Representation

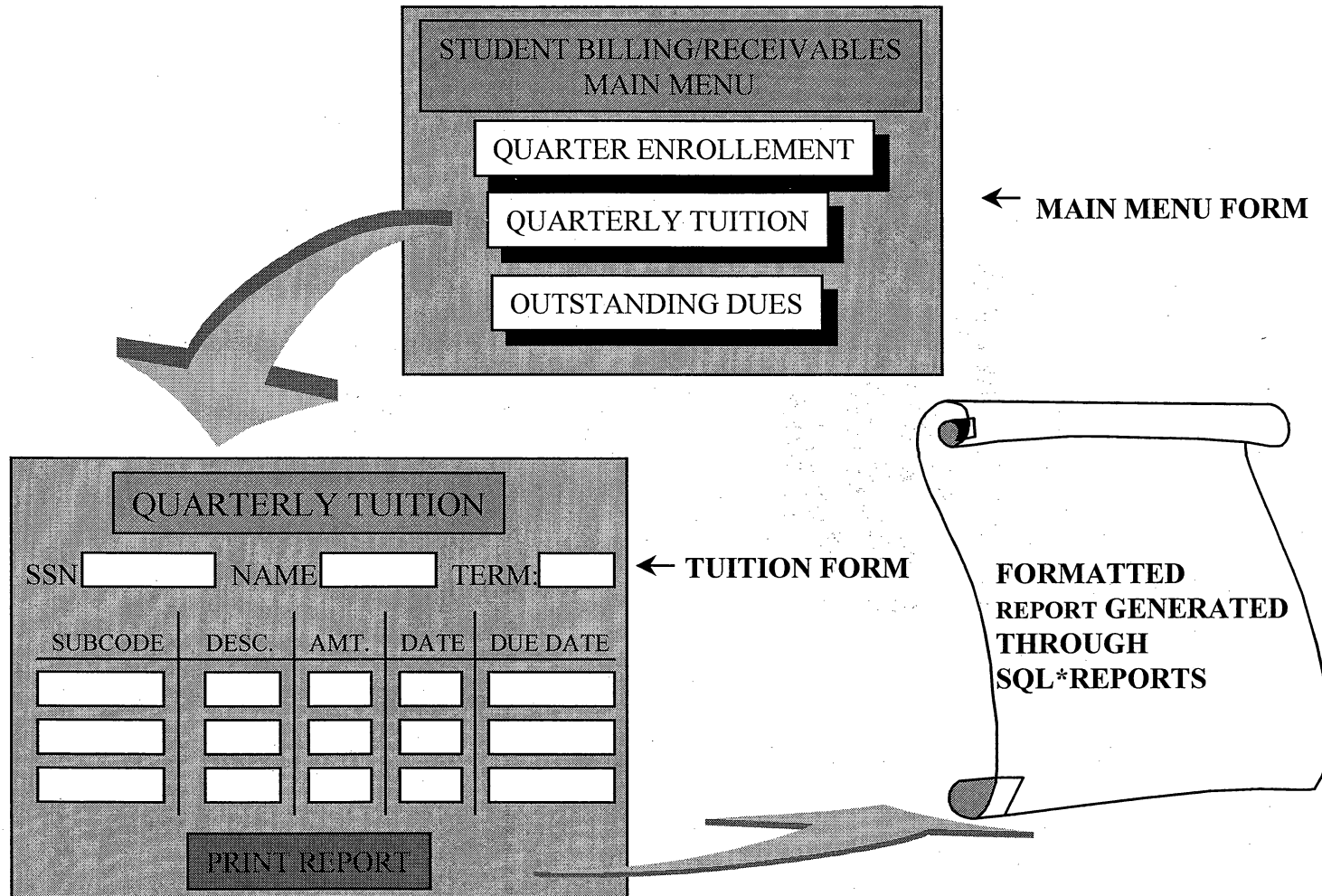
All the data will be represented in ORACLE tables and viewed by means of SQL\*Forms and SQL\*Reports. The entity Student is a relational table. The form 'Students Data' is an independent form derived from the Student base table. There is also a report which prints out all the students enrolled in the University that corresponds to the Students Data form. The report is generated using an SQL statement based on the Student table. This is achieved by selecting the student SSNo and the Name from the Student table.

A sample Main menu, Form, and the corresponding report are shown in Figure 2.2.3.

#### 2.2.4. System Interface Description

This application allows easy ad-hoc access to student billing information and also allows efficient retrieval of student information from the desktop of the end user. The billing information is updated in the mainframe applications. The relevant information will be extracted using FOCUS, a symbolic programming language and loaded into the database tables nightly.

FIGURE 2.2.3. INFORMATION CONTENT REPRESENTATION - EXAMPLE



The final B/R database is a set of ORACLE tables, SQL Forms to view, insert, and update data, and SQL Reports to retrieve often used data. SQL Reports are generated using SQL scripts.

The advantages of this interface are:

- Ease of use (user is informed of the valid choices),
- Minimal entry errors (items can be selected instead of typing; spelling errors can be avoided; and invalid entries can be prevented).

Pick Lists are used for certain fields which have a set of entries. For example, if the attribute 'Extr-ind' can take values Blank, 'C' or 'E' the corresponding pick list gives the description for each of these values.

The B/R module will have three levels of user interaction:

- Ad-hoc queries,
- Routine data entry, and
- Maintenance data entry (updating, deleting).

Once all requirements were collected and analyzed, the

next step was to create the conceptual data model which consists of:

- Identifying entities (e.g., Student, Payment, etc.),
- Determining key attributes of each entity (e.g., Student has entities SID, Name, etc.),
- Establishing relationships and defining constraints (e.g., Billed is the relationship between Student and Payment).

#### 2.2.5. Conceptual Database Design

The design process begins with conceptual design phase, which includes a clear definition of database requirements, content, structure, interrelationships, and constraints. The conceptual design results in a conceptual model (a high-level data model) that possesses the following characteristics:

- a) Expressiveness,
- b) Simplicity,
- c) Minimality,
- d) Diagrammatic representation, and



e) Formality.

Prior to defining the design parameters, it is critical to identify the entities and the information to be recorded about those entities. Therefore, the initial step is to gather and review the database requirements.

2.2.5.1. Requirements and Collection Analysis

In order to design the database effectively, we must know the requirements of the users and the intended uses of the database in as much detail as possible. These include new and existing users and applications. Meetings were held with the accounting staff and the analysts for the following activities:

- Identification of the major application areas and user groups,
- Review of the existing documentation concerning the application, and
- Analysis of the types of transactions and their frequencies.

The following is a detailed summary of the database contents and user requirements:

- Application area: the database should hold and offer information related to the student accounts, the account description, subcodes.
- Application users: the application should be accessible to the accounting staff and also to the analysts in the Administrative Computing Dept.
- Operating environment: the application is currently on UNIX platform (SunOS). The application uses ORACLE Version 7.3 along with its components SQL\*Forms 4.5 and SQL\*Reports 2.5. ORACLE 7.3 was chosen for this application primarily because of its existence in the Administrative and Computing Department at CSUSB and also for its powerful capabilities such as, flexibility to expand the application in the future, enormous data processing capabilities, etc. It is also an industry standard Relational database management system. Multiple users can access the application simultaneously.

#### 2.2.5.2. Development of Entity-Relationship (E-R)

##### Model

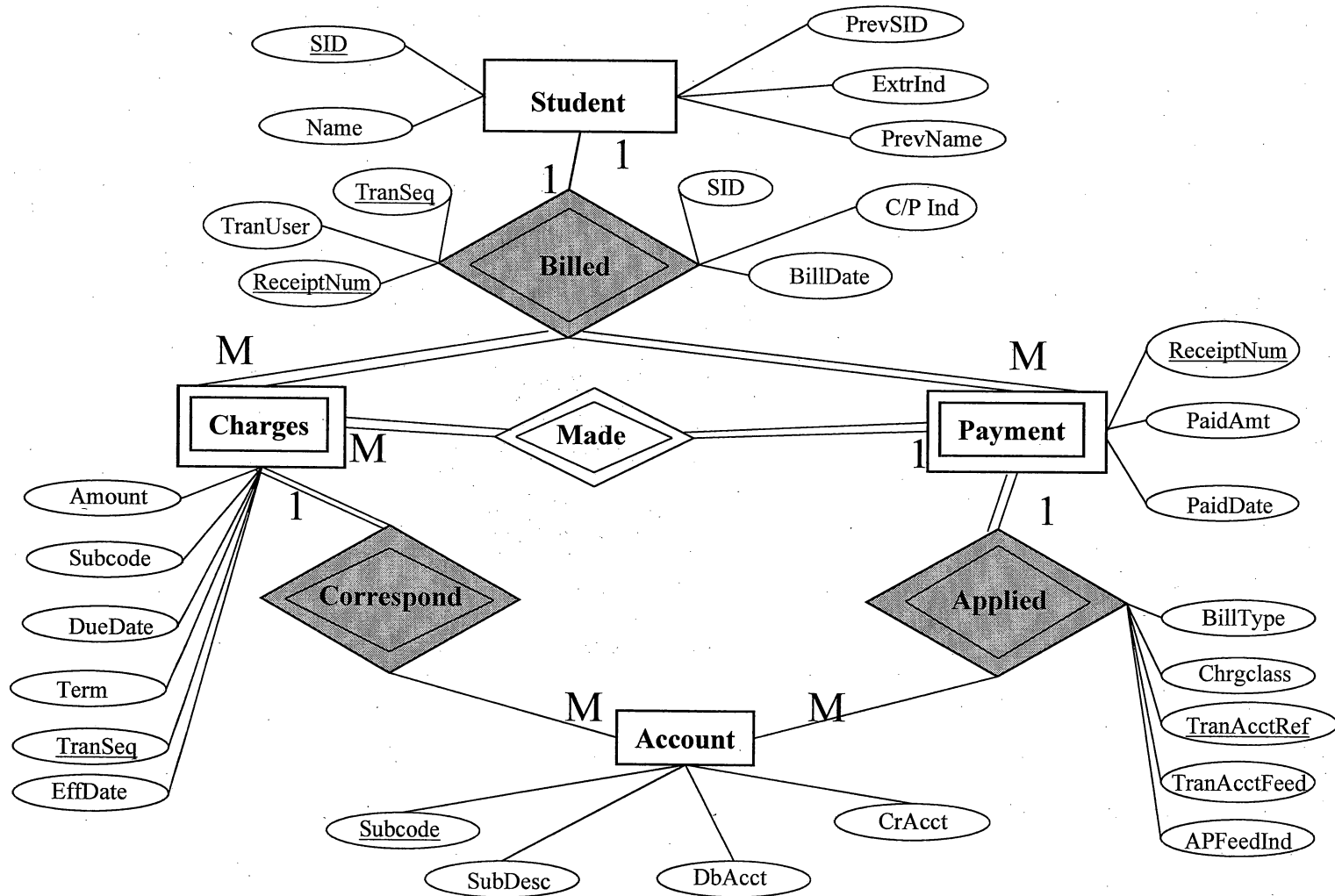
Following the requirements collection and analysis, is the development of the conceptual schema. The conceptual schema is a concise description of the data requirements of

the users and includes detailed description of the data types, relationships, and constraints. The conceptual schema is expressed by means of E-R diagrams. An entity represented in the E-R model is a 'thing' in the real world that has an independent existence. E.g., an entity can be an object with physical existence such as a person or an object with conceptual existence such as a company. Each entity has a specific set of attributes (properties) that describe the entity. Each entity also has an attribute called the primary key attribute and it is used to identify each entity uniquely. The E - R diagram is shown in Figure 2.2.5.2.

#### 2.2.5.2.1. Identify Entities and Associated Attributes

Entities are classified as either regular or weak entity types. A regular entity is also referred to as a strong entity. If an entity's existence is independent of another entity, then it is known as a regular entity. If an entity's existence is dependent on another entity, then the depending entity is called weak entity. In this application, the entities "Student" and "Account" are regular entities and the rest are all weak entities. A regular entity is drawn as a single lined rectangle where as

FIGURE 2.2.5.2. ENTITY RELATIONSHIP DIAGRAM



a weak entity is drawn as a double lined rectangle.

Attributes are a set of properties that define an entity. E.g., the entity "Student" possesses properties such as the social security number, name, etc. Each of these attributes derives their values from a corresponding domain. Attributes can be of several types such as:

- Atomic or Composite: an atomic attribute cannot be divided into sub-properties. Atomic attributes are also called simple attributes. Composite attributes are made up of atomic attributes.
- Single or Multi-valued: a single valued attribute has a single value. A multi-valued attribute has a set of values.
- Base or Derived: a base attribute is an original attribute of an entity where as a derived attribute is a computed value from one or more of the base values or other derived values.

The following are the entities and their corresponding attributes identified in the Billing/Receivables application.

**Student:** This entity specifies the demographic information of a student like the Name, SSNo., ExtrInd., etc. Attribute details are shown in Table 2.2.4.2.1.1.

Table 2.2.4.2.1.1. Attribute Details for Student Entity

Attribute/ Description	Size	Data type	S or M <sup>1</sup>	C or A <sup>2</sup>	B or D <sup>3</sup>
SID/Social Security Number of the student	9	VARCHAR	S	A	B
PrevSID/ Previous Social Security Number of the student if	9	VARCHAR	S	A	B

---

<sup>1</sup> S = Single valued; M = Multi valued

<sup>2</sup> C = Composite; A = Atomic

<sup>3</sup> B = Base; D = Derived

any or else blank					
Name/Name of the student	32	VARCHAR	S	A	B
PrevName/Prev ious name of the student	32	VARCHAR	S	A	B
ExtrInd/Indic a-tor to the type of student	1	VARCHAR	S	A	B

**Charges:** This entity specifies the charges charged under each transaction sequence. The effective dates and the due dates are also included in this entity type. This entity is also a weak entity. The attribute details of this entity type are shown in Table 2.2.4.2.1.2.

Table 2.2.4.2.1.2. Attribute Details for Charges Entity

Attribute/ Description	Size	Data type	S or M	C or A	B or D
Term/Quarter	3	VARCHAR	S	A	B
Subcode/Id of the payment	5	VARCHAR	S	A	B
AMOUNT/AMOUNT CHARGED TO THE STUDENT	9	NUMBER	S	A	B
EffDate/Date when the bill becomes effective	8	DATE	S	A	B
DueDate/Date when the payment is due	8	DATE	S	A	B
TranSeq/ Sequence number of the transaction	1	VARCHAR	S	A	B



**Payment:** This entity holds the details of the payments made by the students. This is also a weak entity. The details of the attributes are shown in Table 2.2.4.2.1.3.

Table 2.2.4.2.1.3. Attribute Details for Payment Entity

Attribute/ Description	Size	Data type	S or M	C or A	B or D
ReceiptNum/ Number on the payment receipt	5	VARCHAR	S	A	B
PaidAmt/Amo- unt of payment	9	NUMBER	S	A	B
PaidDate/Date the payment was made	8	DATE	S	A	B

**Account:** This entity type holds the details of the various types of accounts. This is a strong entity and the details

of this entity type are shown in table 2.2.4.2.1.4.

Table 2.2.4.2.1.4. Attribute Details for Account Entity

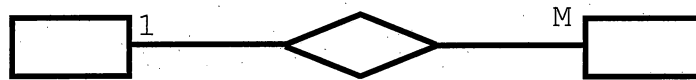
<b>Attribute/ Description</b>	<b>Size</b>	<b>Data type</b>	<b>S or M</b>	<b>C or A</b>	<b>B or D</b>
Subcode	5	VARCHAR	S	A	B
SUBDESC/DESCRIP TION FOR THE SUBCODE	30	VARCHAR	S	A	B
CrAcct/Credit account number of the subcode	15	VARCHAR	S	A	B
DBACCT/DEBIT ACCOUNT NUMBER OF THE SUBCODE	15	VARCHAR	S	A	B

2.2.5.2.2. Identify Relationships and  
Associated Attributes

As described earlier, the Billing/Receivables database has four entities. Each of these entities are associated

among themselves by means of relationships. To set up a relationship between any two entities, one must determine the nature of the relationship. There are three types of relationships as described below.

- One-to-many relationship (1-M): a one-to-many relationship is the most common type of relationship in a relational database. In this type of relationship, a record in Table A (Entity A) can have more than one matching record in Table B (Entity B). However, a record in Table B can have atmost one matching record in Table A.

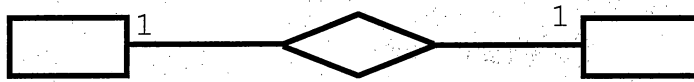


- Many-to-many (M-M): in this type of relationship, a record in Table A can have more than one matching record in Table B, and similarly a record in Table B can have more than one matching record in Table A.



- One-to-one relationship (1-1): this is not a very common relationship. In this type of relationship, a record in Table A can have no more than one matching record in

Table B, and similarly a record in Table B can have no more than one matching record in Table A.



The entities involved in a specific relationship are called the participants of the relationship, and the number of participants in that relationship defines the degree of the relationship. The participation level of an entity in a relationship can either be total or partial. It is said to be total participation when every instance of the participating entity participates in at-least one instance of the relationship, otherwise, the participation level is termed as partial. The following are the relationships that were derived in the Billing/Receivables application.

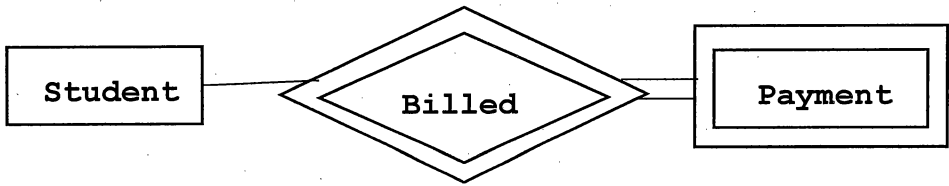
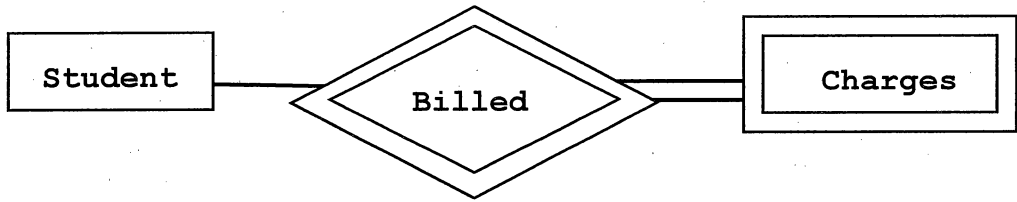
**Billed:** is a one-to-many relationship between Student and Charges and Student and Payment. Every student has a set of charges to pay where as one particular charge can be applied to only one student. Similarly, every student has makes several payments where as one particular payment corresponds to only one student. Student participation in Billed is partial where as Charges and Payment participation is total. Every student might not have charges or Payment but every

charge is billed to a student and every payment is made by a student. Billed is a weak relationship because Charges and Payment are weak entities. The attribute details of this relationship are shown in Table 2.2.4.2.2.1.

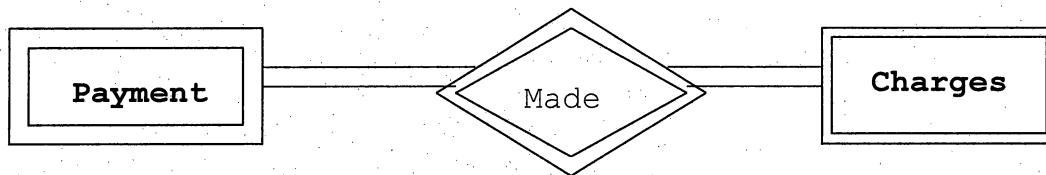
Table 2.2.4.2.2.1. Attribute Details for Billed Relationship

<b>Attribute/ Description</b>	<b>Size</b>	<b>Data type</b>	<b>S or M</b>	<b>C or A</b>	<b>B or D</b>
TranSeq/Id for the transaction being processed	4	VARCHAR	S	A	B
TranUser/Us er id of the person recording the transaction	2	VARCHAR	S	A	B
BillDate/Da te bill prepared	8	DATE	S	A	B

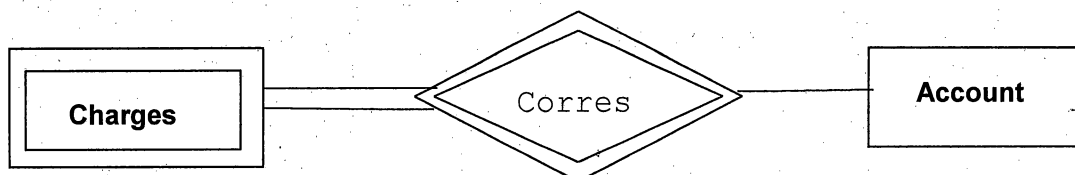
C/Pind/ Indicator for charge or payment	1	VARCHAR	S	A	B
ReceiptNum/ Number on the receipt for the payment	5	VARCHAR	S	A	B



**Made:** is a one-to-many relationship between Payment and Charges. Every Payment is made for many Charges and many charges together are paid at once. Payment participation in Made is total where as Charges participation in Made is partial. Every charge might not have a payment but every payment will have a charge. Made is a weak relationship because both Charges and Payment are weak entities.



**Correspond:** is a one-to-one relationship between Charges and Account. Every charge corresponds to a subcode and every subcode corresponds to just one charge. Charges participation in this relationship is total where as Account participation in this relationship is partial. Every bill is charged based on a subcode but every subcode is not billed to a student. Correspond is a weak relationship because Charges is a weak entity.



**Applied:** is a one-to-many relationship between Payment

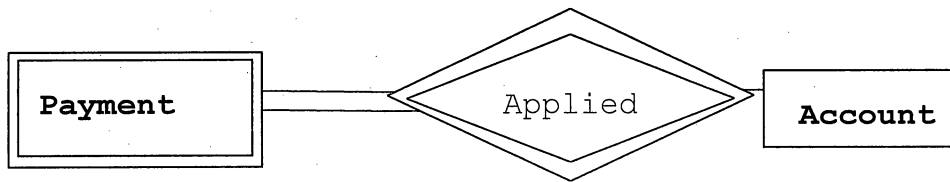
and Account. Every payment constitutes different subcodes in the Account. The participation of Payment in Applied relationship is total where as the participation of Account in Applied relationship is partial. Every payment corresponds to a subcode but every subcode might not have a payment. Applied is a weak relationship because Payment is a weak entity. The attribute details are shown in Table 2.2.4.2.2.2.

Table 2.2.4.2.2.2. Attribute Details for Applied Relationship

<b>Attribute/ Description</b>	<b>Size</b>	<b>Data type</b>	<b>S or M</b>	<b>C or A</b>	<b>B or D</b>
TranAcctRef/ User defined id for the transaction	6	VARCHAR	S	A	B
TranAcctFeed/ Whether transaction has been fed	1	VARCHAR	S	A	B



to the accounting system					
APFeedInd/ Indicates transaction in the Account Payable	1	VARCHAR	S	A	B
BillType/Spec ifies if it is registration, tuition, etc.	1	VARCHAR	S	A	B
ChrgClass/Or- der in which the payment has to be applied	3	VARCHAR	S	A	B



#### 2.2.5.3. DERIVATION OF FUNCTIONAL DEPENDENCIES

Once the E-R diagram is mapped with the database requirements and constraints, the next logical step is to derive functional dependencies (FDs). Functional dependencies are used to group attributes into relational schemas that are in normal form. A functional dependency is a constraint between two sets of attributes from a database.

In a relation schema  $R$ ,  $X$  functionally determines  $Y$  if and only if whenever two tuples of  $R$  agree on their  $X$ -value, they must necessarily agree on their  $Y$ -value or, alternatively, the values of  $X$  component of a tuple uniquely or functionally determine the values of the  $Y$  component.

Symbolically, this relation is represented as  $X \rightarrow Y$ .

Based on the review of the user requirements, and the E-R diagram, the following FDs have been established. The functional dependencies are grouped by each determinant attribute.

SID -----> PrevSID

SID -----> Name  
SID -----> PrevName  
SID -----> ExtrInd  
Tran\_Seq -----> TranUser  
Tran\_Seq -----> Term  
Subcode -----> SubDesc  
Subcode -----> CrAcct  
Subcode -----> DbAcct  
Tran\_Seq -----> BillDate  
Tran\_Seq -----> EffDate  
Tran\_Seq -----> DueDate  
Tran\_Seq -----> ReceiptNum  
Tran\_Seq -----> TranAcctRef  
Tran\_Seq -----> ChrgClass  
Tran\_Seq -----> BillType

#### 2.2.5.4. Development Of Base Relations

This section defines the initial set of base relations that can be derived from the E-R diagram. A relation is a mathematical term for a table. A relation consists of tuples and fields. A tuple corresponds to a row and a field corresponds to a column which is also an attribute. The number of tuples in a relation is the cardinality of the relation. The number of attributes in a relation is the degree of the relation. In every relation there is a primary key which uniquely identifies the tuples. There could also be a foreign key which corresponds to a primary key in the base relation. Primary key is denoted with a solid underline and foreign key is denoted with a dashed underline.

The following base relations have been derived for the Billing/Receivables database

##### **Student**

<u>SID</u>	PrevSID	Name	PrevName	ExtrInd
------------	---------	------	----------	---------

##### **Billed**

<u>TranSeq</u>	TranUser	BillDate	C/Pind	ReceiptNum
----------------	----------	----------	--------	------------

### **Charges**

SID   Term   Subcode   Amount   EffDate   DueDate   TranSeq

### **Payment**

SID            TranSeq            ReceiptNum            PaidAmt

PaidDate

### **Applied**

BillType            TranAcctRef            TranAcctFeed

APFeedInd

ChrgClass

### **Account**

Subcode            SubDesc            CrAcct            DbAcct

## 2.2.5.5.            Normalization of Base Relations

The next step is to normalize the above base relations. Normalization of data is a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties. Unsatisfactory relation schemas are those schemas which have redundant attributes, i.e., composite and multi valued attributes. Normalization

is achieved through three steps which are First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF). Hence, a database is desired to be at least in Third Normal Form.

First normal form is defined to disallow multi-valued attributes, composite attributes, and their combinations. It states that the domains of the attributes must include only atomic values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. The only attribute values permitted by 1NF are single atomic values from a domain of such values. As per the definition of 1NF, we see that the Billing/Receivables database is already in the 1NF since it doesn't contain any multi-valued or composite attributes.

Second normal form depends on the concept of full functional dependency. A functional dependency  $X \twoheadrightarrow Y$  is fully functional dependent if removal of any attribute A from X means that the dependency doesn't hold any more. A relation schema R is in 2NF if every non primary key attribute in R is fully functionally dependent on the primary key of R. As per the definition of 2NF, the

Billing/Receivables database is already in 2NF since all the non primary key attributes are fully functionally dependent on the primary key of the base relation.

Third normal form is based on the concept of transitive dependency. A functional dependency  $X \twoheadrightarrow Y$  is a transitive dependency if there is a set of attributes  $Z$  that is not a subset of any key of the relation  $R$ , and both  $X \twoheadrightarrow Z$  and  $Z \twoheadrightarrow Y$  hold. A relation schema  $R$  is in 3NF if every non primary key attribute of  $R$  is transitively dependent on the primary key. As per the definition of 3NF, the Billing/Receivables database is already in 3NF because there are no transitive dependencies.

So the base relations are the normalized relations.

#### 2.2.5.6. Database Implementation

Created relational tables and query scripts using ORACLE 7.3, SQL\*Forms 4.5, and SQL\*Reports 2.5.

#### 2.2.5.7. Testing and Validation

The Billing/Receivables application has been validated for several validation criteria. Various classes of tests have been conducted to verify the following database functionality:

- a) Correctness: the result of correctly framed queries should return all matching records.
  - b) Multi-user functionality: tests for ORACLE's multi-user features, e.g. concurrent read access. This hasn't yet been tested as the database hasn't yet been loaded in the system.
- The extraction of data from the mainframe system is done using FOCUS. FOCUS is a symbolic programming language. It is mostly used for commercial applications. FOCUS is best able to handle complex input/ output functions where a large volume of data needs to be processed at one time. The extraction of data was done by Mr. Ted Schiffer of Administrative Department.

### 2.3. SYSTEM REFERENCE

The proposed database development will employ Sun OS 4.3.1 operating system. The database management system(DBMS) used will be Oracle 7.3, along with its components SQL\*Plus 3.1, SQL\* Forms 4.5, and SQL\*Reports

### 2.5.

### 2.4. SOFTWARE PROJECT CONSTRAINTS

Personnel - The system is being designed and implemented



by a graduate student, Sushma Lukalapu under the guidance of advisors, Dr. Josephine Mendoza, CSCI Dept. and Ms. Lorraine Frost, Administrative Computing Services.

## 2.5. FUNCTIONAL DESCRIPTION

### 2.5.1. Functional Description

#### 2.5.1.1. Processing Narrative

The information about student accounts is available to the accounting staff so they can create ad-hoc queries to retrieve the data. Also, some data can be obtained from the reports created in this project. The SQL reports provided to the accounting staff are:

- names of all students who have been charged a particular subcode by term,
- listing of subcodes charged to a particular student by term or transaction date, and
- detailed description of the fees paid by a student for a term.

The result of each of the above queries will be a SQL report which may be viewed onscreen or printed.

#### 2.5.1.2. Restrictions/Limitations

- a) Accounting staff will have a few reports to choose from (as mentioned above). Any other reports for which SQL scripts are not written could be created using ad-hoc queries.
- b) The staff might receive some data which may not fit into the existing schema.

#### 2.5.1.3. Performance Requirements

The size of the database is relatively small ( Seven base tables with an average of six fields per table) when compared to ORACLE's processing capacity. Since, it is only sample data, not more than ten records are inserted in a table. So the response time is expected to be good. However, there are some implicit requirements for this system design :

##### a) User-friendly

- Shortest time possible required for the user to get used to the database.
- Sample forms are graphically illustrated, and menu-driven oriented pull down menus.

#### b) Testability

- Each requirement in this design has been verified and thoroughly tested .

#### 2.5.1.4. Design Constraints

Constraints like data constraints, domain constraints, and enterprise constraints will be taken into account:

- a) The enterprise constraints mean the set of specifications initially defined by the enterprise. The B/R module doesn't have any enterprise constraints.
- b) The domain constraints specify the range of values for a certain attribute (e.g., In the Student table of the B/R module, the attribute Extr\_ind can take values -Blank, C or E).
- c) Data constraints specify the size of each attribute(e.g., In the Student table of the B/R module, the attribute SSN has to be 9 digits).

#### 2.5.1.5. Supporting Diagrams

ER diagrams that provide an overview of the database, DDLs (Data definition language) used to create the tables and the final Oracle tables comprise the supporting

diagrams.

## 2.6. BEHAVIORAL DESCRIPTION

### 2.6.1. System State

The database state should be consistent, which means that the database should satisfy all the state constraints such as:

- a) Attribute key constraint which specifies whether an attribute or a set of attributes is a unique attribute of an entity type. e.g. Subcode is a unique key for the entity 'Account'.
- b) Attribute structural constraint which specifies whether an attribute is single valued or multi-valued and whether or not null is allowed for the attribute. There are no multi-valued attributes in the Billing/Receivables module.

### 2.6.2. Events and Actions

It will be ensured that the proposed database will maintain the integrity constraints:

- a) Entity integrity will ensure that every primary key is not null. e.g. SSNo. cannot take a null value.

b) Referential integrity makes sure that every foreign key that exists (is not null) in a child table refers to an existing primary key in the parent table. Referential integrity could be violated when updating a relation. There are three basic update operations on relations:

- Inserting a tuple: When inserting a tuple in a child table, referential integrity can be violated if that same value does not exist in the parent table. This can be corrected by changing the value in the child table to some valid value that exists in the parent table. It can also be corrected by inserting a new tuple with this value in the parent table (this insertion has to be acceptable). e.g. Inserting a student and a corresponding charge in the 'Charges' table without inserting the student (student's SSNo.) in the 'Student' table will violate referential integrity.
- Deleting a tuple: If referential integrity is violated while deleting a record, it could be corrected either by rejecting the deletion, cascade the deletion by deleting tuples that reference the tuple being deleted or by modifying the referencing attribute values that cause the violation; each such value is set to null or changed to reference another valid tuple. However, if the

referencing attribute that causes the violation is part of the primary key, then it cannot be set to null , because then it would violate entity integrity. e.g. Deleting a student's SSNo. from the 'Student' table when the SSNo. exists in the 'Charges' table violates referential integrity.

- Modifying the value of an attribute: If referential integrity is violated while modifying the value of a primary key, then it can be corrected by the above two methods because modifying is similar to deleting a tuple and inserting another in its place. However, if a foreign key is modified, then the new value should refer to an existing tuple in the referenced relation.

Also whenever the primary key is updated, the corresponding foreign key also has to be updated. If a primary key is deleted, the corresponding foreign key should also be deleted. If a new record is inserted into the child table, the record has to exist in the parent table. Table 2.6.2 shows the primary keys and the foreign keys for the B/R module.

The performance is expected to be good considering that the size of the database (only seven tables) is small. Some factors could affect the response time. For example, the B/R database will communicate with the server via the campus computer network, which at times, could be overloaded. And also other factors outside the application, like adding new fields to the tables, adding additional constraints, etc., may affect the performance and these are beyond the scope of the project.

#### 2.7.1. Performance Bounds

#### 2.7. VALIDATION CRITERIA

PRIMARY KEYS	PRIMARY KEY TABLE	FOREIGN KEY TABLE
SID	STUDENT	CHARGES, PAYMENT
TRANSEQ	BILLED	CHARGES, PAYMENT
SUBCODE	ACCOUNT	CHARGES
TRANACCTREF	APPLIEDTO	NO REFERENCE
RECEIPTNUM	PAYMENT, BILLED	NO REFERENCE

Table 2.6.2. Primary and Foreign Keys

### 2.7.2. Classes of Tests

The result of correct ad-hoc queries will be all matching records. Three specific tests will include:

- a) Unit Test: Individual tables will be tested for the check constraints, unique key constraints and the entity constraints.
- b) Integration Test: All tables together will be tested for referential integrity constraints.
- c) System Test: The whole database will be tested to ensure that it performs all the allocated functions.

### 2.7.3. Expected Software Response

As discussed earlier, B/R will be designed to have good response time.

### 2.7.4. Special Considerations

- a) New data entries may contain information which wasn't anticipated in the existing schema. So, the database might need periodic modifications.
- b) Users might want some enhancements once the B/R module is in daily use.



### **3. PHYSICAL DATABASE DESIGN**

The physical design phase of the database is the physical implementation of the conceptual design using a Relational Database Management System like ORACLE 7.3. This phase defines the data storage structures, associated mappings, and access paths related to the database application. This process generally involves the design of tables, loading of data into the tables, design of forms, generating reports, and testing the performance. The following section describes the physical design in detail.

#### **3.1. CREATION OF TABLES**

The tables for the Billing/Receivables database were created using ORACLE 7.3. The normalized relations from the conceptual design were converted into the table format using the 'create table' command in ORACLE. There are seven tables in the Billing/Receivables database. Each table has on average six fields.

The syntax for the create table command is shown in the Appendix A.

The order in which the tables are dropped and created is critical because of the referential integrity

constraints. The sequence of the tables in the Billing/Receivables database is as given below.

```
drop table Student;
```

```
drop table Account;
```

```
drop table Billed
```

```
drop table Charges;
```

```
drop table Payment;
```

```
drop table Applied;
```

**Create table Student(**

```
    SID      VARCHAR2(9) constraint student_sid_pk  
PRIMARY KEY,
```

```
    PrevSID  VARCHAR2(9) constraint student_prevsid_uk  
UNIQUE,
```

```
    Name     VARCHAR2(32) constraint student_name_nn NOT  
NULL,
```

```
    PrevName  VARCHAR2(32),
```

```
    ExtrInd  VARCHAR2(1) constraint student_extrind_ck
```

```
        check (Extr_Ind in ('-', 'C', 'E')));
```

**Create table Billed(**

```
TranSeq      VARCHAR2(4) constraint billed_transeq_pk
PRIMARY KEY,

TranUser      VARCHAR2(2) constraint billed_tranuser_nn
NOT NULL,

BillDate      DATE,

C/Pind        VARCHAR2(1) constraint billed_cpind_ck check
              (C/Pind in ('C', 'P')),

ReceiptNum    VARCHAR2(5) constraint billed_recptnum_pk
              PRIMARY KEY);
```

**Create table Account(**

```
Subcode      VARCHAR2(5) constraint account_subcode_pk
              PRIMARY KEY,

SubDesc       VARCHAR2(30) constraint account_subdesc_nn
NOT NULL,

CrAcct        VARCHAR2(15) constraint account_cracct_uk
UNIQUE,

DbAcct        VARCHAR2(15) constraint account_dbacct_uk
UNIQUE);
```

**Create table Charges(**

```
SID          VARCHAR2(9) constraint charges_sid_fk
              references Student ON DELETE CASCADE,

EffDate      DATE,

DueDate      DATE,

Term         VARCHAR2(3),

Subcode      VARCHAR2(5) constraint charges_subcode_fk
              references Account ON DELETE CASCADE,

Amount       NUMBER(9,2),

TranSeq      VARCHAR2(4) constraint charges_transeq_fk
              references Billed ON DELETE CASCADE);
```

**Create table Payment(**

```
SID          VARCHAR2(9) constraint payment_sids_fk
              references Student ON DELETE CASCADE,

PaidDate     DATE,

PaidAmt      NUMBER(9,2),

ReceiptNum   VARCHAR2(5) constraint
payment_recptnum_fk references Billed ON DELETE
CASCADE);
```

### Create table Applied(

```
    TranAcctRef    VARCHAR2(6) constraint applied_ref_pk
PRIMARY KEY,

    TranAcctFeed   VARCHAR2(1),

    APFeedInd      VARCHAR2(1),

    ChrgClass      VARCHAR2(3),

    BillType       VARCHAR2(1) constraint
charges_billtype_ck

                check (BillType in
('I','T','N','Q','F','B','H','C','S','E','W','D','A'))
);
```

### 3.2. DATA LOADING

The data extraction from the file management system was done using FOCUS. FOCUS is a data retrieval language. The data was extracted by Mr. Ted Schiffer of the Administrative Computing Services. This data was to be loaded into the ORACLE tables using SQL\*Load. This task could not be completed because SQL\*Loader was not accessible at school (it wasn't loaded in the system). Sample data has been

loaded and the application has been tested with the sample data. All the queries have been reported to return all matching records.

### 3.3. DESIGN OF SQL\*FORMS

SQL\*Forms were designed for data entry and updating. SQL\*Forms aid in the quick development of form based applications for entering, querying, updating, and deleting data. Before going further into detailed discussion of forms, it is important to understand the following terms:

**Block:** a section of form that corresponds to a table in the database. Blocks provide a simple mechanism for grouping related items into a functional unit for storing, displaying, and manipulating records.

**Multi-record block:** a block that can display more than one record at a time.

**Record:** data from one row in base table as represented in a form.

**Base table:** a table in the database on which a block is based.

**Base table item:** an item that directly relates to the base

table.

**Control item:** these items are populated with database values using SQL statements. e.g. TotalAmt is obtained by adding up all the AmtCharged from the Charges table.

**List item:** a list of text elements that can be displayed as pop list sometimes called drop list. e.g. BillType in the Charges table is a list item.

**Radio groups:** display a fixed no. of options that are mutually exclusive. Each option is represented by an individual radio button. e.g. APFeedInd in the Applied table.

**Text items:** default data type for text items is CHAR.

Eleven forms have been developed for the Billing/Receivables application. These forms can be grouped under two different categories namely Independent forms, Master-detail forms.

The independent forms directly correspond to the base tables wherein data can be entered and updated. Data for some fields (e.g. Extr Ind, C/P Ind, AP Feed Ind, Acctg Feed) in the independent forms can be entered through list items and radio buttons. All the base tables have the

corresponding independent forms. Examples of independent forms are Student, Charges, Payment, etc.

Master-detail form on the other hand is an association between two base table blocks, the master block and the detail block. The relationship between the master block and the detail block reflects a primary key to foreign key relationship between the tables on which the blocks are based. The master block is related to the detail block through the Join condition. The Join condition establishes the primary key item in the master block and the foreign key item in the detail block. The data in the master-detail forms cannot be updated. They are just used for displaying data.

The master-detail forms are Student-Charges; Account-Charges; Student-Charges-Payment.

Steps to create and modify forms are shown in Appendix A.

### 3.4. DESIGN OF SQL\*REPORTS

A report is a summary of information that is well organized and formatted to suit the user's specifications. Reports provide more flexibility in presenting data that is easy to understand. Some daily use examples of reports are



mailing labels, invoices, sales summaries, etc.

For the Billing/Receivables application several reports can be generated based on query criteria. These reports could be generated on single-variable or multi-variable requests. Single-variable requests are:

- Search for students based on a subcode,
- Search for subcodes charged for a student, etc.

Multi-variable requests are:

- Search for students based on subcode by term or transaction date,
- Search for subcodes per student by term or transaction date, etc.

The reports consist of several SQL queries. Every base table has a corresponding report (e.g. Student data, charges for students, payment by a student, list of the subcodes along with their description, etc.). There are three multi-variable request reports (e.g. Search for students based on subcode by term, Search for subcodes per student by term, charges for a student, etc.).

Steps to generate and modify reports are shown in

## Appendix A.

### 3.5. FUTURE ENHANCEMENTS

- Transaction User can have a pick list that shows the user name corresponding to the two digit code.
- Accounting feed is defined as a radio button. That can be changed to a pop list.
- Have a Main menu for the reports also.
- Within the main menu for reports, have a sub menu for the students report, in the increasing order of the SSNo, alphabetical order, and according to the external indicator.
- Have several other reports like querying charges by transaction sequence, charges by term, etc. in the main menu.

#### **4. DATABASE VALIDATION**

Once the database is successfully designed and implemented, the next step is to validate the database application for its functionality. Positive results in these validation tests would imply that the database would perform satisfactorily and initial objectives have been met. Tests performed on the Billing/Receivables application can be grouped under three categories:

- Unit testing,
- Integration testing, and
- System testing.

Test results under each of these categories is presented in the following sections.

##### **4.1. UNIT TESTING**

Unit testing focuses on the verification effort of the smallest unit of the database application. For the B/R application, each form is tested individually for its functionality like proper storage and retrieval of data, etc.

Similarly, each report is also tested to verify data organization, format, and accurate data retrieval of the user requested query.

#### 4.2. INTEGRATION TESTING

Following the unit testing phase, the next phase would focus on the performance of the integrated components of the database application. The individual components like forms and reports are integrated using SQL\*Forms. Various tests were conducted to verify the linkage between all forms and reports.

#### 4.3. SYSTEM TESTING

In this phase of testing, the B/R application is tested as a complete system. Two primary tests were conducted to ensure the completeness of the system which include tests on database user-friendliness and multi-user feature of ORACLE.

User-friendliness: tests are performed to verify whether the application prompts the user for information in an accurate and easily interpretable format. Also to verify if useful help text was provided when necessary.

Muti-user feature: tests are performed to verify the muti-user feature of ORACLE like the concurrent read access.

## **APPENDIX A: CREATING TABLES, FORMS AND REPORTS**

## SYNTAX TO CREATE TABLES IN ORACLE

```
create table table_name(  
    column1 datatype(size)  
column_constraint|table_constraint,  
    column2 datatype(size),  
    ..... );
```

The syntax used to create table is not case sensitive. The create table command enforces different kinds of constraints on the table including primary keys, foreign keys, not nulls, and check conditions. Enforcement of these constraints allow ORACLE to maintain the database integrity. Based on the type of the data, the datatype is specified as CHAR, VARCHAR, DATE, NUMBER, etc. The advantage of using VARCHAR over CHAR is that VARCHAR uses only the required spaces where as CHAR uses all defined spaces by filling the unfilled spaces (after using the required spaces) with blanks. A majority of the fields in the Billing/Receivables database are VARCHAR. The dates are given the datatype DATE and the amounts are given the datatype NUMBER.

## STEPS TO CREATE FORMS IN ORACLE

- After invoking the Oracle Forms designer, choose File -> Open -> Form from the menu bar.
- Once the list of files are listed, Select -> stucgpmt.fmb
- After the file is selected, Forms returns to the Object Navigator. Define a Window and a Canvas View to place the stucgpmt form on. Give a name to the Window and the Canvas View.
- Next, go to the tool bar and open the layout editor.
- Once the form module is created, the default name can be changed from the property sheet. Other properties in the property sheet could also be changed.

### Steps to create blocks

- From the object navigator, select the blocks node by clicking on it one time
- Click on the create icon on the vertical tool bar
- Connect to the database from File -> Connect

- This will bring up the New Block dialog box. Specify the name of the base table, name of the block, and name of the canvas if an independent form has to be created. If a master detail form has to be created, then specify the name of the master table and also the name of the detail table along with the Join condition to relate to these two tables. Select columns under Items. New Block Options allows to select only the wanted columns. Select the style of the block as Tabular or Form.
- From the tools menu in the Object Navigator, select the layout editor to view the skeletal form of the block just created.

#### Steps to Test the Forms

Generate the form to create an executable version of the form and then run the form. Select the <Generate> option from the File menu. Execute the form by selecting <Execute> from the file menu. Then select the <Run> button from the tool bar. The form that was just created appears. The form is tested by inserting a new record, query an existing record, update the queried record, querying the record just inserted. To insert a new record, press the <Insert Record>, enter the data and then select <save>.



### Steps to Modify the Forms

The form could be modified by going into the Layout Editor. The blocks can be repositioned. The fields in a block can be selected or deselected by opening the appropriate block. The fields could be re-named by opening the properties sheet for that particular field. List items, Radio buttons, Pop lists, etc. can be used to define a field.

### **STEPS TO CREATE REPORTS IN ORACLE**

- Invoke Oracle Reports by double-clicking on the Oracle Reports Designer.
- Connect to the database through File → Connect.
- Double-click on the icon in the Data Model Node and the Data Model Editor is displayed.
- Select the query tool by clicking on it once in the Tool Palette.
- Move the mouse pointer into the Data Model editor and click once. A query object represented by a rounded rectangle appears.
- Double click on the query object to display its property

sheet.

- In the SELECT Statement field, type in the select statement for the required data.
- Select OK to close the Query property sheet. A default group containing a list of the selected columns is created.
- Select Tools → Default Layout.
- Select File → Save to save the report.

#### Steps to Test the Reports

- Select File → Run and Oracle Reports displays the Runtime Parameter Form.
- Select Run Report from the Runtime Parameter Form to accept the default values.

The report is tested to see if the desired data has been displayed.

#### Steps to Modify the Reports

- Open the Data Model Editor and double click on the query box.

- Select statements can be written here or the existing select statements can be modified by having various conditions in the Where clause.

## **APPENDIX B: STEPS TO GENERATE A MASTER DETAIL FORM**

Below are the steps to create Student-Charges-Payment master detail form:

After invoking the Oracle Forms designer,

1. Select File->Open->stuchgpmt.fmb from the menu bar.
2. After the file is selected, Forms returns to the Object Navigator. Now define a window and a canvas-view to place the stuchgpmt form.
3. Select New Block option from the Tools in the tool bar. This will bring up the New Block dialog box. Give the name for the new block as Student . Give the name for the window and the canvas view as stuchgpmt.
4. Select the table Student and select Name, SSNo., and Extr Ind columns under items from the table.
5. Select the style of the block as form.
6. Repeat step three for the Is\_Billed block and select the columns SSNO. and Tran Seq. Place this block also on the stuchgpmt canvas.
7. Select the master-detail option and give the join condition between the Student block and the Is\_Billed block. SSno. joins the two tables. Name this block as

Billed.

8. Select the form layout for the Billed block.
9. Repeat step six for the Charges block. Select the columns Tran Seq, Term, Subcode, Tran Amt, Eff Date, Due Date, Chrg Class and place the block on stuchgpmt canvas.
10. Select the master detail option and give the join condition between the Billed block and the Charges block. Tran Seq joins the two blocks. Name this block as Charges.
11. Select the tabular layout for the Charges block.
12. Create a new block for the Payment table and place it on the stuchgpmt canvas.
13. Select the columns Receipt Num, Tran Seq, Amt Paid, and Date Paid .
14. Join the charges block and the Payment block by the Tran Seq and name it as payment.
15. Select the form layout for the payment block.
16. Once all the required blocks are created, go to the layout editor in the tool bar and view the skeletal

form of the blocks just created.

17. Now, we can double click on each item of the block and go to its property sheet. The blocks properties can also be changed by double clicking on the entire block.

18. The items and the blocks on the form can be moved within the form to the desired location.

19. Select the Button \_Palette block under the stuchgpmt form.

20. Select the UP, DOWN, QUERY, SAVE items under the Button\_Palette block.

21. Write a When\_New\_Form\_Instance trigger under the form level trigger to query the form, as you enter the form. The trigger is shown in the Appendix D.

22. Have another item in the stuchgpmt under the Student block and name it as MAINMENU. Write an item level trigger to return to the mainmenu on double clicking the item. The trigger is shown in the Appendix D.

23. To run the form, select File->Generate and then File->Execute.

24. The form is automatically queried and all the information is in appropriate fields.

25. To enter new data in the form, go to the last record and do scroll down. Enter the data and save it. Then do query on it. New data is stored.



---

APPENDIX C: FORMS CREATED FOR PROJECT

The Object Navigator for the stuchgpmt form is shown in Figures. 1, 2, and 3.

Figure 1

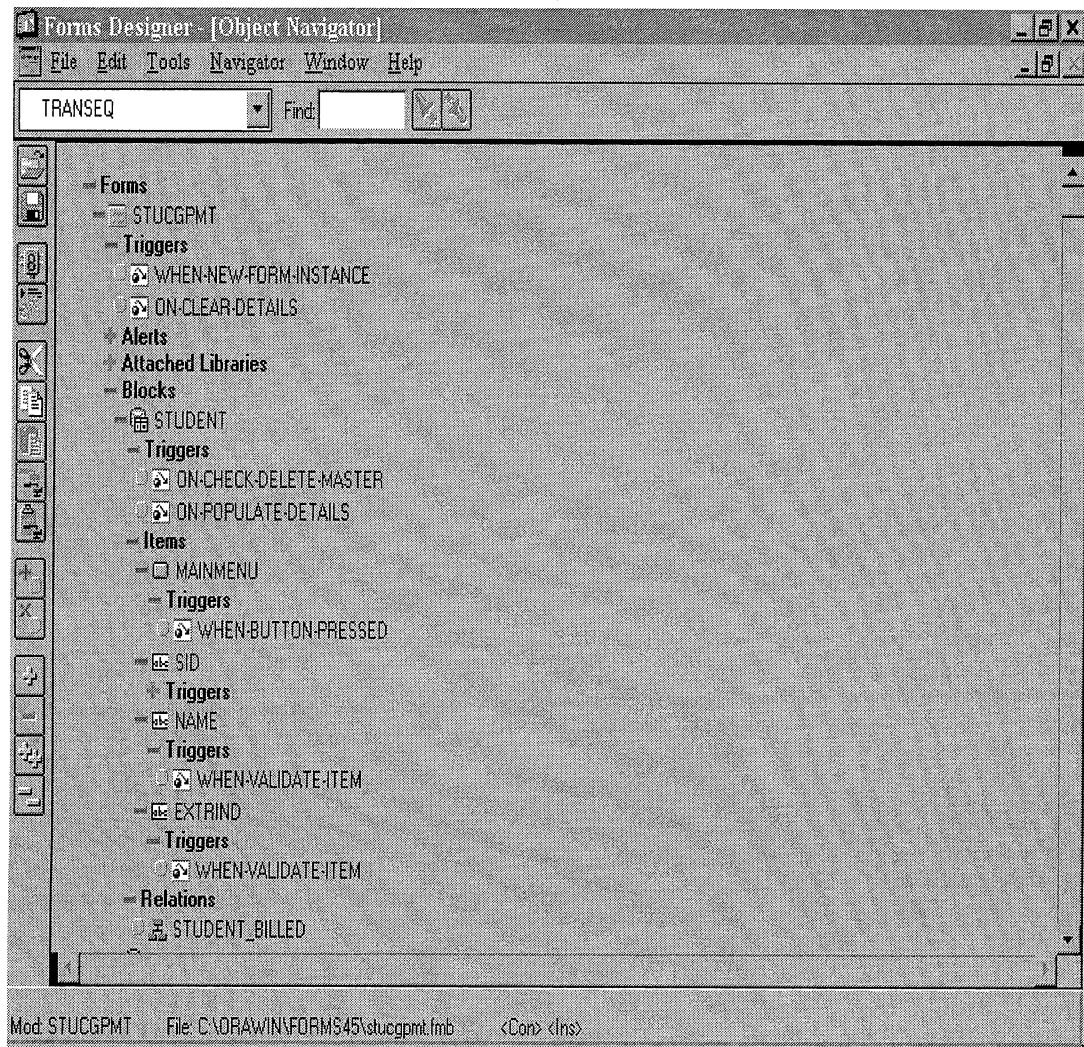


Figure 2

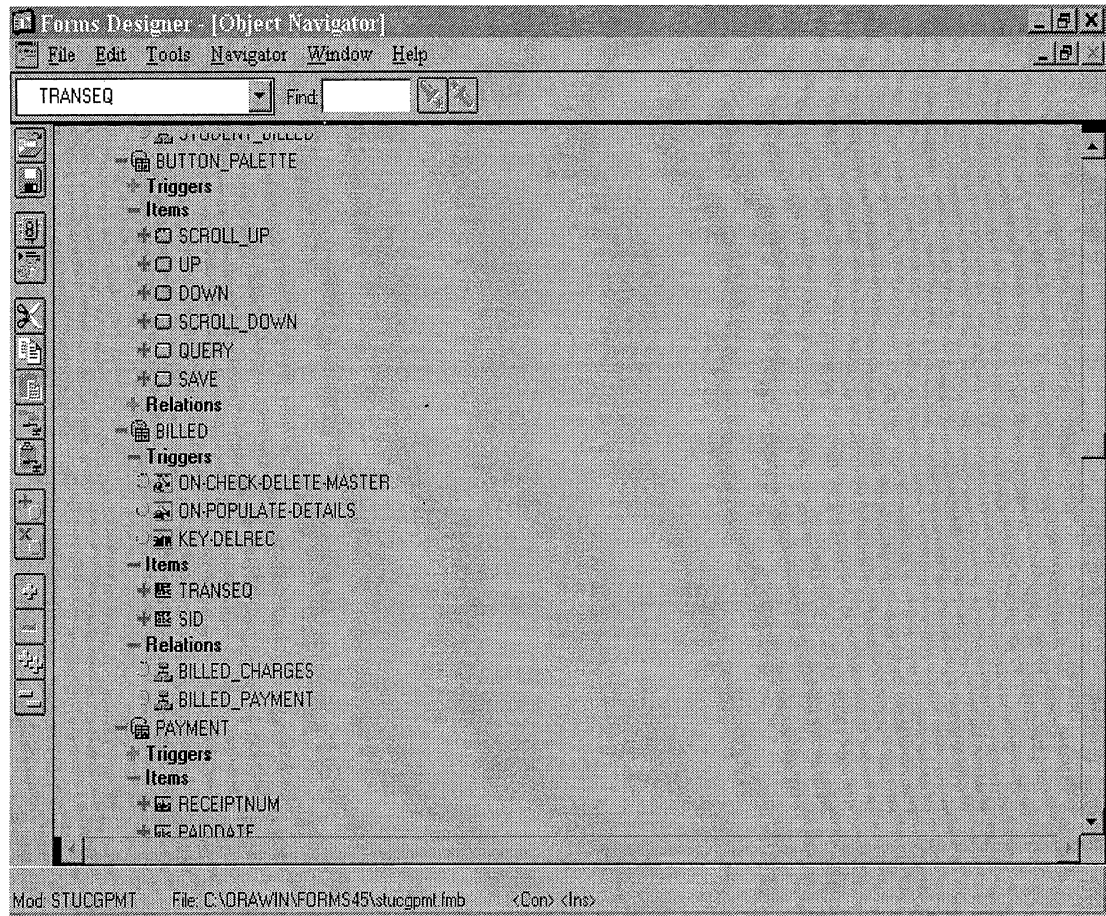
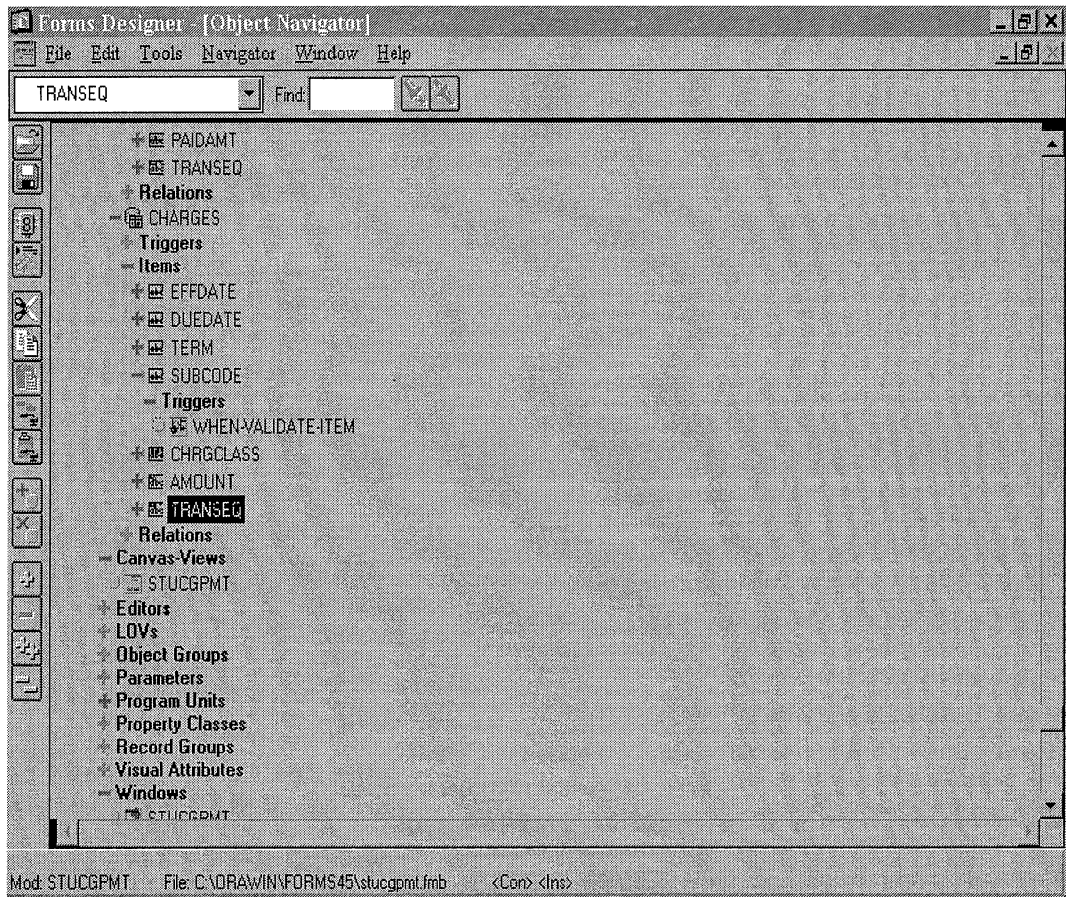
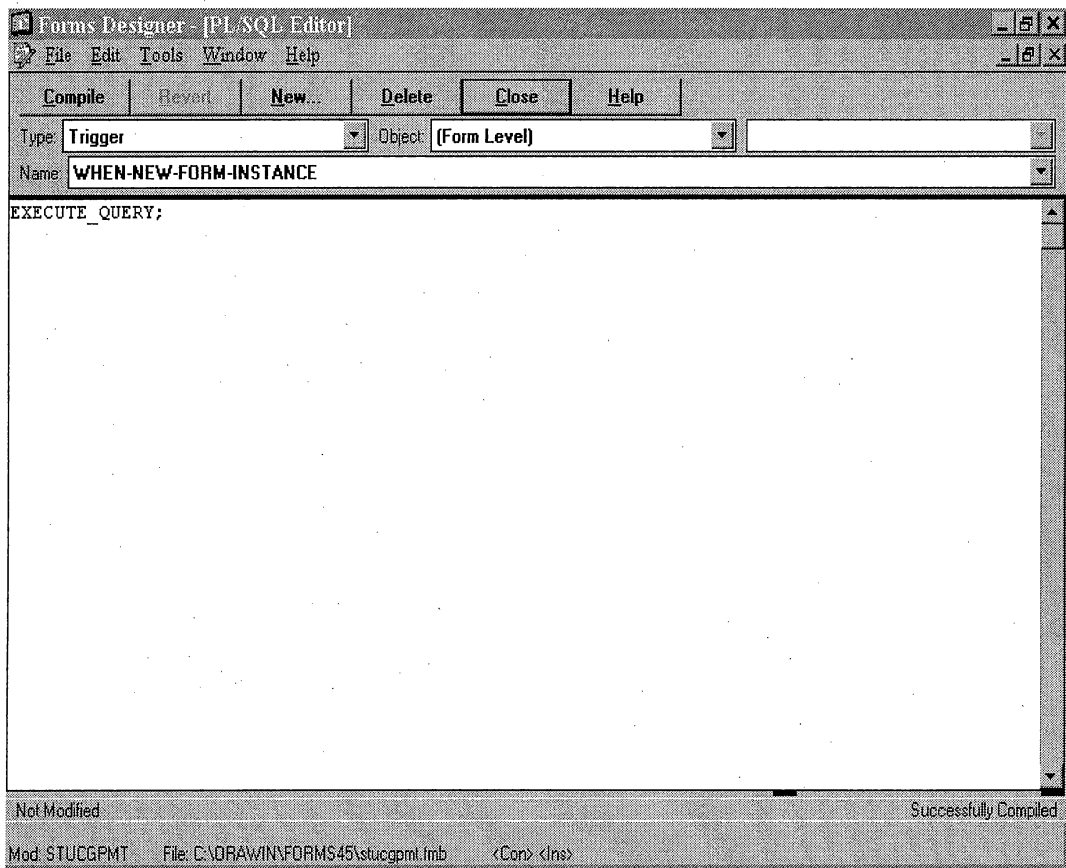


Figure 3



A form level trigger is written to execute the query (the data is entered in the appropriate fields as the form is opened) as we enter the form. A form level trigger for the stuchgpmt form is shown in Figure 4.

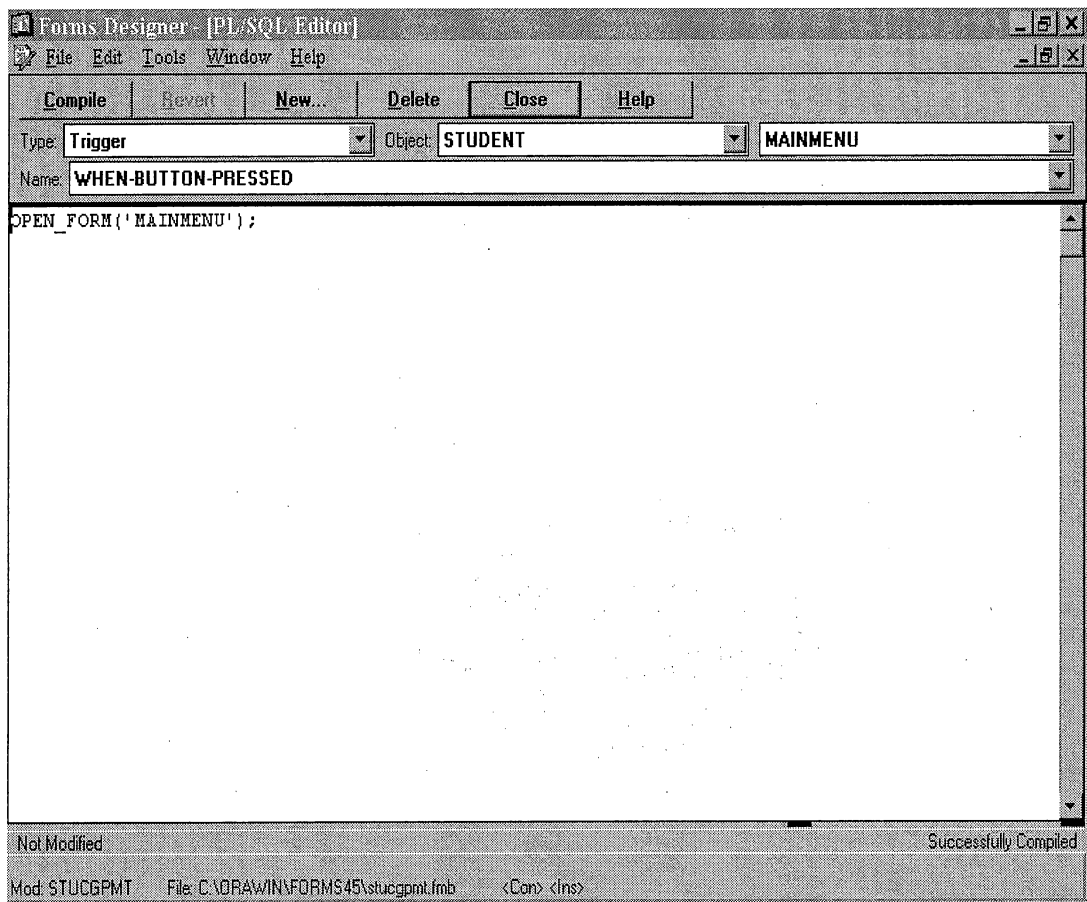
Figure 4



Item level trigger is written for an item action to take place. Main menu is an item in the student block. To return to the main menu from the stuchgpmt form, SQL editor is

opened. The code is shown in Figure 5.

Figure 5

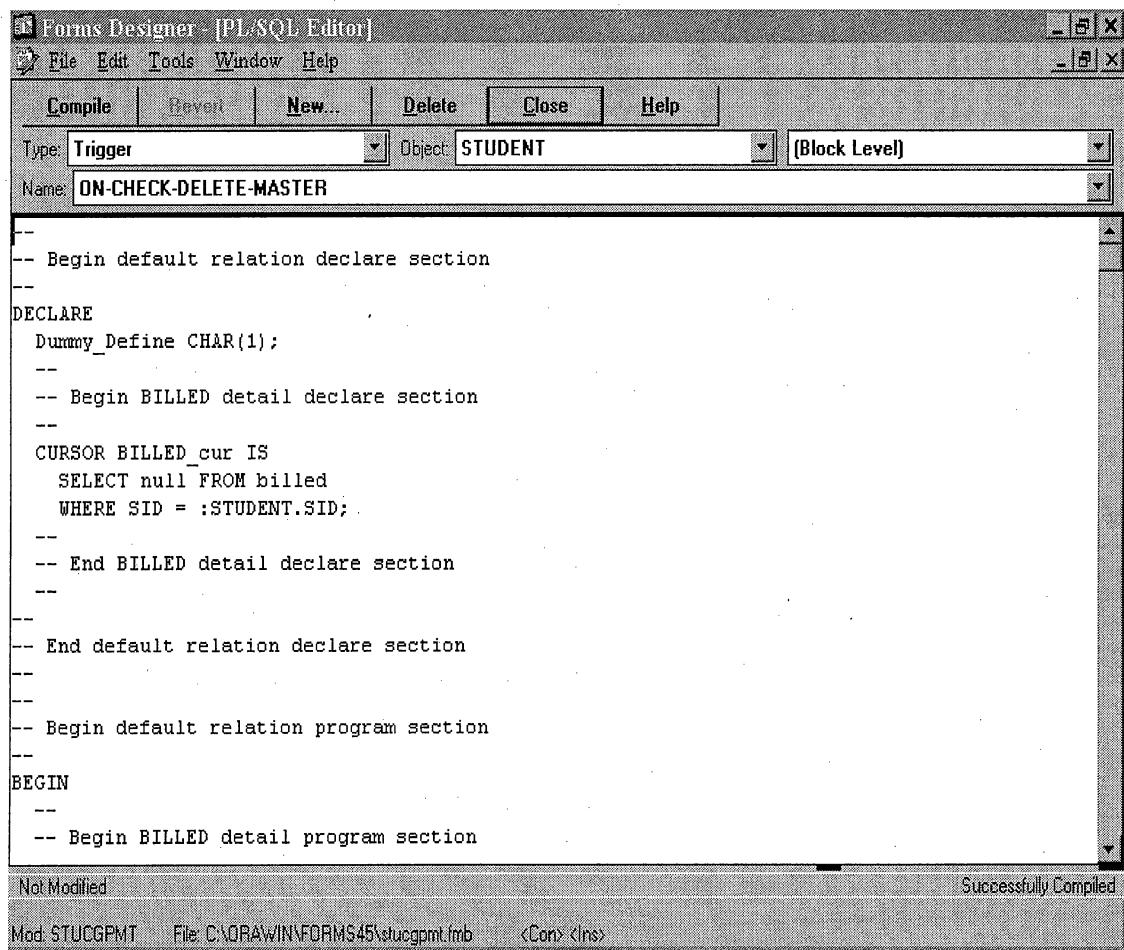


Block level trigger is executed when a block is opened.

This trigger takes care of the

table constraints. Block level trigger for the stuchgpmt is shown in Figure 6.

Figure 6.





The main menu appears as shown in Figure 7. The user can select a form from the given choices for data entry.

Figure 7

WELCOME TO THE BILLING/RECEIVABLES MODULE CALIFORNIA STATE UNIVERSITY SAN BERNARDINO	
SELECT ONE OF THE FOLLOWING CHOICES :	
<b>Students</b>	LIST OF ALL ENROLLED STUDENTS
<b>Transactions</b>	TRANSACTION USERS OF EACH TRANSACTION SEQUENCE
<b>Charges</b>	CHARGES FOR EACH TRANSACTION SEQUENCE BY STUDENT
<b>Receipt Nos.</b>	LIST OF RECEIPT NUMBERS OF ALL PAYMENTS
<b>Payments</b>	PAYMENTS FOR EACH TRANSACTION SEQUENCE
<b>C/P Details</b>	DETAILS OF THE CHARGE/PAYMENT APPLIED TO AN ACCOUNT
<b>Subcodes</b>	LIST OF SUBCODES AND THEIR DESCRIPTION
<b>Charges By Term</b>	LIST OF CHARGES FOR A STUDENT BY TERM
<b>C/P By Trans. Seq.</b>	LIST OF CHARGES & PAYMENT BY TRANSACTION SEQUENCE
<b>Students/Subcode/Term</b>	STUDENT LISTING BY SUBCODE BY TERM
<b>Subcodes By Student</b>	SUBCODE LISTING PER STUDENT



The following form shows all the students enrolled in the University.

Figure 8

STUDENTS ENROLLED IN THE UNIVERSITY						
Social Security No.	612701267	Previous Social Security No.	612701267			
Name	sushma	Previous Name	sushma			
External Indicator	REGULAR					
<div>&lt;&lt; &lt; &gt; &gt;&gt; Query Save Mainmenu</div>						
<div>Student_Report</div>						

The following form shows all the details of a charge billed to a student.

Figure 9

<b>CHARGES</b>			
Transaction Sequence	6127	Effective Date	15-MAY-9E
Due Date	15-JUN-98	Term	982
Subcode	11111	Bill Type	Tuition Income Clearing ▼
Charge Class	A ▼	Amount Charged	400
Social Security No.	612701267	C/P Indicator	▼
<div>&lt;&lt;   &lt;   &gt;   &gt;&gt;   Query   Save   Mainmenu</div>			
<div>Charges Report</div>			

The following form shows the transaction details for a charge.

Figure 10

TRANSACTION USERS FOR CHARGES BILLED						
Social Security No.	612701267		Transaction Sequence	6127		
Transaction User	61		Bill Date	15-APR-98		
<<	<	>	>>	Query	Save	Mainmenu
<div>Billed_Report</div>						

The following form shows the payment receipt numbers paid by a student.

Figure 11

PAYMENT RECEIPT NUMBERS			
Social Security No.		Receipt No.	
▲	612701267	01267	
	017725073	25073	
	123456789	56789	
	124356789	57689	
▼	123546789	56879	

<<	<	>	>>	Query	Save	Main Menu
----	---	---	----	-------	------	-----------

The following form checks if the charge or payment has been fed to the accounting system.

Figure 12

CHARGE OR PAYMENT APPLIED TO THE ACCOUNT							
Transaction Account Reference		612701		Transaction Sequence		6127	
Transaction Accounting Feed	<input checked="" type="radio"/>	NOT DONE		Ap Feed Indicator		<input type="radio"/> NO REFUND	
	<input type="radio"/>	FIRST PHASE DONE				<input checked="" type="radio"/> REFUND	
	<input type="radio"/>	CHECK WRITTEN FOR REFUND					
<<		<		>		>>	
Query		Save		Mainmenu			
APPLIED REPORT							

The following form shows all the subcodes and their description.

Figure 13

LIST OF SUBCODES AND THEIR DESCRIPTION							
Subcode	66666		Credit Account	666660000			
Subcode Description	PAYMENT		Debit Account	666660000			
<<	<	>	>>	Query	Save	Mainmenu	Account_Report



Figure 14 is a master detail form that shows all the student names that have been charged a particular subcode on a certain transaction date.

Figure 14

**STUDENTS PER SUBCODE BY TRANSACTION DATE**

Subcode **11111** Debit Account No. **1111100002**

Subcode Description **TUITION** Credit Account No. **1111100001**

Social Security No.	Name	Amount Charged	Amount Paid	Date Paid
612701267	sushma	400	400	01-JUN-98

SC994\_Report
<<
<
>
>>
Query
Save
Mainmenu

Figure 15 is a master detail form that shows all the student names that have been charged a particular subcode by term.

Figure 15

**STUDENT CHARGES-ORDERED BY TERM**

Student Name

Social Security No.

**CHARGES**

Transaction Sequence	Effective Date	Due Date	Subcode	Amount Charged	Term
6127	15-MAY-98	15-JUN-98	11111	400	982
6128	15-MAY-98	15-JUN-98	33333	25	982
6127	15-MAY-98	15-JUN-98	22222	789	982



Figure 16 is a master detail form that shows the names of students that have been charged a subcode per term.

Figure 16

**STUDENTS CHARGES BY SUBCODE**

**SUBCODE**

Subcode  Debit Account No.

Subcode Description  Credit Account No.

**STUDENTS**

Social Security No.	Name	Amount Charged	Amount Paid	Term
612701267	sushma	400	400	982

Figure 17 shows the student charges and payment by term.

Figure 17

**STUDENT CHARGES AND PAYMENT BY TERM**

Social Security No. **612701267**      Name **sushma**      External Indicator **-**

**PAYMENT**

Transaction Sequence **6127**      Receipt No. **01267**      Date Paid **01-JUN-98**      Total Amount Paid **400**

↓

**CHARGES**

Effective Date	Due Date	Term	Subcode	Charge Class	Amount Charged
15-MAY-98	15-JUN-98	982	11111	A	400
15-MAY-98	15-JUN-98	982	22222	A	789

<<    <    >    >>    Query    Save    **Mainmenu**

Figure 18 shows the details of a payment made by a student.

Figure 18

RECORD OF PAYMENTS						
Social Security No.		612701267				
Receipt No.		01267		Date Paid		01-JUN-98
Transaction Sequence				Amount Paid		400
<<	<	>	>>	Query	Save	Mainmenu
Payment Report						

**APPENDIX D: REPORTS CREATED FOR PROJECT**

---

## SUBCODES REPORT

CSU - San Bernardino

## SUBCODES REPORT

Page 2 of 2

Current Date: 03-JUN-99

<u>SUBCODE</u>	<u>SUBCODE DESCRIPTION</u>	<u>CREDIT ACCOUNT</u>	<u>DEBIT ACCOUNT</u>
11111	TUITION	1111100001	1111100002
22222	NON RESIDENT TUITION	2222200001	2222200002
33333	REGISTRATION FEE	3333300001	3333300002
44444	HOUSING	4444400001	4444400002
55555	PARKING	5555500001	5555500002
66666	PAYMENT	6666600001	6666600002

The following is the SQL Code used to generate the Subcodes report:

```
SELECT * FROM Account
```

```
Order by Subcode;
```

## APPLIED TRANSACTIONS REPORT

CSU- San Bernardino  
Date : 03-JUN-99

### APPLIED TRANSACTIONS

Page 2 of 2

<u>Trans Seq</u>	<u>AP Feed Indicator</u>	<u>Trans Acct Feed</u>	<u>Trans Acct Ref</u>
6127	Y	-	612701
0177	N	-	017725
6128	N	-	612801

The following is the SQL Code used to generate the transactions report:

```
SELECT * FROM APPLIED;
```

## Billed Report

CSU- San Bernardino  
Date : 03-JUN-99

### Billed Report

Page 2 of 2

<u>Trans User</u>	<u>SSN.</u>	<u>Trans Seq</u>	<u>Bill Date</u>
01	017725073	0176	17-APR-98
01	017725073	0177	15-APR-98
12	123456789	1234	15-APR-98
14	123546789	1235	15-APR-98
12	124356789	1243	15-APR-98
61	612701267	6127	15-APR-98
01	612701267	6128	17-APR-98

The following is the SQL Code used to generate the Billed report:

```
SELECT * FROM Billed;
```

## Charges Report

CSU - San Bernardino

Charges Report

Page 2 of 2

Current Date: 03-JUN-99

<u>SSNo.</u>	<u>Trans Seq</u>	<u>Subcode</u>	<u>Charge Class</u>	<u>Bill Type</u>	<u>Term</u>	<u>Amt. Charged</u>	<u>Eff Date</u>	<u>Due</u>
017725073	0176	33333	A	Q	974	25	10-MAY-98	11-J
	0177	22222	A	T	982	450	15-MAY-98	15-J
123456789	1234	22222	A	T	982	600	15-MAY-98	15-J
612701267	6127	11111	A	I	982	400	15-MAY-98	15-J
		22222	A		982	789	15-MAY-98	15-J
	6128	33333	AB	Q	982	25	15-MAY-98	15-J

The following is the SQL Code used to generate the Charges report:

```
SELECT * FROM Charges
```

```
Order by SID;
```



## Payments Report

CSU - San Bernardino

## Payments Report

Page 2 of 2

Current Date: 03-JUN-99

<u>SSNo.</u>	<u>Tran. Seq</u>	<u>Receipt No.</u>	<u>Subcode</u>	<u>Amt. Paid</u>	<u>Date Paid</u>
017725073	0176	25074	33333	20	01-JUN-98
	0177	25073	22222	450	01-JUN-98
123456789	1234	56789	22222	550	12-MAR-98
612701267	6127	01267	11111	400	01-JUN-98
	6128	01268	33333	25	01-JUN-98

The following is the SQL Code used to generate the Payment report:

```
SELECT * FROM Payment
```

```
Order by SID;
```

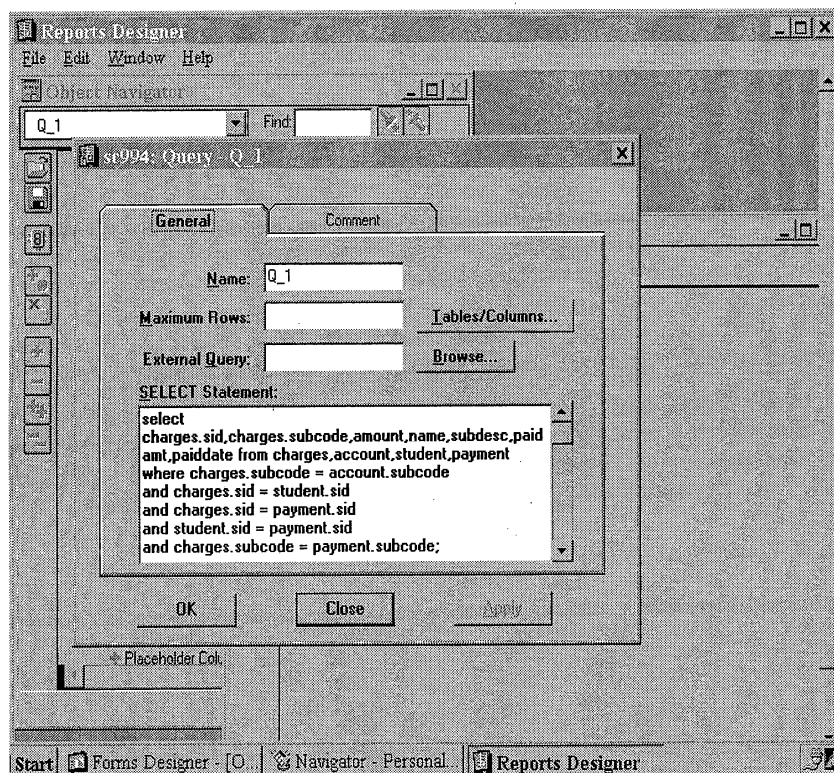
## Subcodes By Transaction Report

CSU - San Bernardino SUBCODES BY TRANSACTION DATE Page 1 of 1

Date: 03-JUN-99

SSNo.	Extr. Ind.	Subcode	Chrg. Class	Amt. Charged	Amt. Paid
017725073	E	22222	A	450	450
017725073	E	33333	A	25	20
123456789	E	22222	A	600	550
612701267	-	11111	A	400	400
612701267	-	33333	AB	25	25

The SQL code for the above report is shown below.



CSU - San Bernardino STUDENTS PER SUBCODE LISTING Page 1 of 1

Date: 03-JUN-99

SSNo.	Name	Subcode	Subcode Desc.	Amt.. Charged	Amt.Paid	Tran. Date
017725073	ramana	22222	NON RESIDENT TUIT:	450	450	01-JUN-98
017725073	ramana	33333	REGISTRATION FEE	25	20	01-JUN-98
123456789	radha	22222	NON RESIDENT TUIT:	600	550	12-MAR-98
612701267	sushma	11111	TUITION	400	400	01-JUN-98
612701267	sushma	33333	REGISTRATION FEE	25	25	01-JUN-98

The SQL code used to generate the above report is shown below:

Reports Designer - [sc400: Query - Q\_1]

File Edit Window Help

General Comment

Name: Q\_1

Maximum Rows: Tables/Columns...

External Query: Browse...

SELECT Statement:

```
select charges.subcode, charges.transeq, account.subdesc, amount, effdate, billdate,
paiddate
from charges, account, billed, student, payment
where charges.subcode = account.subcode
and payment.subcode = charges.subcode
and payment.subcode = account.subcode
and billed.transeq = charges.transeq
and billed.transeq = payment.transeq
and charges.transeq = payment.transeq
and payment.sid = charges.sid
and student.sid = charges.sid
and student.sid = payment.sid
and term = '982'
and student.sid = '612701267';
```

OK Close

Start Forms Designer - [O...] Navigator - Personal Reports Designer ...

## Students Report

CSU - San Bernardino

### Students Report

Page 2 of 2

Current Date: 03-JUN-99

SSNo.	Name	Previous SSNo.	Previous Name	Extr. Ind.
612701267	sushma	612701267	sushma	-
017725073	ramana	017725073	ramana	E
123456789	radha	123456789	radha	E
124356789	uma	124356789	uma	-
123546789	divya	123546789	divya	-
111111111	venkat	111111111	venkat	C

The SQL Code used to generate the Students report is shown below:

```
SELECT * FROM Student;
```

If we order by name in the select statement, the names are displayed in the ascending alphabetical order.

## BIBLIOGRAPHY

- [1]Batini, C., Ceri, S., and Shamkant N., Conceptual Database Design - An Entity Relationship Approach, The Benjamin/ Cummings Publishing Company Inc., 1992.
- [2]Connolly, T., Begg, C., and Strachan, A., Database Systems - A Practical Approach to Design, Implementation and Management, Addison Wesley Publishing Company Inc., 1996.
- [3]Date, C. J., An Introduction to Database Systems, 6th ed., Addison Wesley Publishing Company Inc., 1995.
- [4]Elmasri, R., and Shamkant, N., Fundamentals of Database Systems, The Benjamin/Cummings Publishing Company Inc., 1989.
- [5]Koch, G., ORACLE7 - the complete reference, McGraw-Hill, Inc., 1993.
- [6]Luers, T., Essential Oracle 7, First ed., Sams publishing Inc., 1995.
- [7]Oracle 7.3 Manuals, Oracle Corporation, USA, 1996.
- [8]Ricardo, C., Database Systems - Principles, Design and Implementation, Macmillan Publishing Company, 1990
- [9]Rolland, Relational Database Management with Oracle, 2<sup>nd</sup> ed., Addison Wesley Publishing Company Inc., 1992.