Public Library Database

Jeremy Martinez

Team 9

Table of Contents

Phase 1

1.1 Fact Finding Techniques and Information Gathering

1.1.1 Fact-Finding Techniques

Through a former business relationship and resources that could be found online, we gathered information on public libraries. One member of our group formerly worked at a library and has personal experiences with how a library functions. However, information that we may not be familiar with will be checked through websites of numerous public libraries to find common qualities that would be necessary for a database. For the purpose of finding information about how upper management works in a public library we will be interviewing a head librarian at a local library.

1.1.2 Introduction to Enterprise

Libraries are public businesses comprised of head librarians and library technicians. Some library technicians are also volunteers. Head librarians manage schedules of library technicians and order new books for the library. Library technicians help clients with checking out books, renting a study room, and also organize books that are new or returned. Volunteers are limited to just organizing books.

1.1.3 Structure of the Enterprise

For this database, head librarians are assigned to a specific library and oversee the operations of that library. Head librarians are assigned based on their education and if a library is in need of a head library. Head librarian oversee library technicians who are in charge of organizing books and fulfilling requests of cardholders who want to check out materials or rent a study room. Materials are divided into categories that separate themselves from each other, such as electronic or print. Study rooms are facilities within the library that can be found on separate levels and for separate activities. Cardholders can interact with library technicians in order to request a material check out or reserve a study room.

Library technicians can be categorized as being paid, or volunteers. Volunteers are in charge of organizing books as well as placing returned books in their correct positions within the library.

1.1.4 Enterprise Data

The ones who enter the data are the librarians. The librarians update the records of available books and media during checkouts. For returns, the librarians make sure the book is returned in an acceptable condition and update the status of the availability of the media. Librarians must also input any new or donated items into the database. Late fees are automatically calculated by the current date and the due date. The system keeps track of all fees, however, it is the individual librarian who is responsible to receive the fees from the cardholder who has a late fee on their account and update the info.

1.1.5 Conceptual Database

There are multiple jobs in the enterprise, technicians that fulfill the requests of cardholders, volunteers that organize books, head librarians that order new books, technicians that keep track of study rooms. Our database seeks to cover most of the daily operations a library would consist of on any given day.

1.1.6 Major Entities and Relationship Sets

A major entity in this database would be employees. This class holds general information for all employees such as SSN, name, address, phone number, and there will be additional information in other subclasses for more specific jobs.

Another major entity is the cardholders. The cardholder entity contains basic information about library card holders that is different from employees, such as rental fees, and card identification number.

The last major entity in our database would be the library entity. This entity has information for

the location of the library and the identification number for the library.

A common relationship is the requests relationship. This relationship connects the materials, study rooms, cardholders, and library technicians.

1.2 Conceptual Database Design

1.2.1 Entity Set Descriptions

Library

Description: Stores basic information about locations of libraries such as address, state, etc. A new entry is only added when a new library is built in a city. Some of the general information for a library can be changed, such as phone number, or ID number.

Primary Key: ID

Candidate Keys: ID

Address

Strong Entity:

Fields to be Indexed: ID

Address

Attribute Name	ID	Address	State	Zipcode	City	Phone
Description	ID for internal Database	Street Address of Library	State that library is located	Zip code of library	City of library	Phone number of library
Domain/Type	Integer	String	String	Integer	String	Integer
Value-Range	0-Max	Any	Any	0-Max	Any	0-Max
Default Value	None	None	None	None	None	None
Null Allowed	No	No	No	No	No	No
Unique	Yes	Yes	No	No	No	Yes
Single/Multivalue	Single	Single	Single	Single	Single	Single

Simple/Composite	Simple	Composite	Simple	Simple	Simple	Simple

Employee

Description: Stores basic information for all employees in a given library, like name or SSN. A new entry added when a new employee is hired at a library. Much of the information found in this entity can be edited, such as address or phone number.

Primary Key:

ID

Candidate Keys:

ID

Name

Strong Entity

Fields to be Indexed:

ID

Name

Attribute Name	ID	LName	FName	SSN	Address	State	Phone	Email	Bdate	Library ID	Payrate	Position
Description	ID for internal Database	Employee last name	Employee first name	Employee SSN	Employee Address	Employee State	Employee phone numb	Employee email	Employee Birth date	ID of library of work	Money earned by hours worked	Position of employee at work

							er					
Domain/ Type	Integer	String	Strin g	Integ er	String	String	Integ er	Strin g	Intege r	Integer	Integer	String
Value-Range	0-Max	Any	Any	0-Max	Any	Any	0-Max	Any	0-Max	0-Max	0-Max	Any
Default Value	None	None	None	None	None	None	None	None	None	None	None	None
Null Allowed	No	No	No	No	No	No	No	No	No	No	No	No
Unique	Yes	No	No	Yes	No	No	No	Yes	No	Yes	Yes	No
Single/ Multivalue	Single	Single	Singl e	Singl e	Single	Single	Singl e	Singl e	Singl e	Single	Single	Single
Simple/ Composite	Simple	Simpl e	Simp le	Simp le	Simpl e	Simple	Simp le	Simp le	Simpl e	Simple	Simple	Simple

WorkLog

Description: This entity is related to the library technician in that it logs all the work done by both paid and non-paid employees. This includes the date worked, hours worked, and the time started.

Primary Key:

Date

Candidate Keys:

Date

Time

Strong Entity

Fields to be Indexed:

Date

Time

Attribute Name	Date	Time	Hours	EmployeeID
Description	Date that employee worked	Time that employee started working	Hours that employee worked	ID of Employee filling out form
Domain/Type	Integer	Integer	Integer	Integer

Value-Range	0-Max	0-Max	0-Max	0-Max
Default Value	None	None	None	None
Null Allowed	No	No	No	No
Unique	No	No	No	Yes
Single/ Multivalue	Single	Single	Single	Single
Simple/ Composite	Simple	Simple	Simple	Simple

Material

Description: This entity is a superclass that contains the general information for the materials being kept within the library. This contains the title, year published, and genre, and the item identification number.

Primary Key:

Title

Candidate Keys:

Title

Strong Entity

Fields to be Indexed:

Title

Attribute Name	Title	Year	Genre
Description	Item's given title	Year item was published	Genre that item falls into
Domain/Type	String	Integer	String
Value-Range	Any	0-Max	Any

Default Value	None	None	None
Null Allowed	No	No	No
Unique	No	No	No
Single/ Multivalue	Multivalue	Single	Single
Simple/ Composite	Composite	Simple	Composite

Print

Description: This entity includes all items that could be printed and the information that comes with that. This entity contains attributes such as ISBN, publisher, and issue date.

Primary Key:

ID

Candidate Keys:

ID

ISBN

Strong Entity

Fields to be Indexed:

ID

ISBN

Attribute Name	ISBN	Publisher	IssueDate	NumCopy	ID	LibraryID	Author
Description	ISBN of printed	Publisher of printed item	Date the printed	The number of	ID of item	ID to show what library	Author of text

	item		item was issued	copies available		item belongs to	
Domain/Type	Integer	String	Integer	Integer	Integer	Integer	String
Value-Range	0-Max	Any	0-Max	0-Max	0-Max	0-Max	Any
Default Value	None	None	None	None	None	None	None
Null Allowed	No	No	No	Yes	No	No	No
Unique	Yes	No	No	No	Yes	Yes	No
Single/ Multivalue	Single	Single	Single	Single	Single	Single	Multivalue
Simple/ Composite	Simple	Simple	Simple	Simple	Simple	Simple	Simple

Electronic

Description: This entity contains the information that would be exclusive to electronic materials found in the library. The name of the production company, whether or not is visual, and the amount of copies on hand is held here.

Primary Key:

ID

Candidate Keys

ID

Strong Entity

Fields to be Indexed:

ID

Attribute Name	Production	Visual	NumCopy	ID	Library ID
Description	Production	Whether the	Number of	ID of item	ID that item

	company of the material	material is visual or not	copies on hand of the material		belongs to
Domain/Type	String	Boolean	Integer	Integer	Integer
Value-Range	Any	True/False	0-Max	0-Max	0-Max
Default Value	None	None	None	None	None
Null Allowed	No	No	Yes	No	No
Unique	No	No	No	Yes	Yes
Single/ Multivalue	Multivalue	Single	Single	Single	Single
Simple/ Composite	Composite	Simple	Simple	Simple	Simple

StudyRooms

Description: The information stored in this entity has descriptions for the study rooms found within libraries that members can rent. The attributes found in this entity includes capacity, purpose, and location.

Primary Key:

ID

Candidate Keys:

ID

Weak Entity

Fields to be Indexed:

ID

Attribute Name	Capacity	Purpose	Location	ID	LibraryID

Description	Capacity of a given study room	Purpose or activities done in a given room	Location of the study room within the library	ID of study room in library	Library ID that hosts study room
Domain/Type	Integer	String	String	Integer	Integer
Value-Range	0-Max	Any	Any	0-Max	0-Max
Default Value	None	None	None	None	None
Null Allowed	No	No	No	No	No
Unique	No	No	Yes	Yes	Yes
Single/ Multivalue	Single	Single	Single	Single	Single
Simple/ Composite	Simple	Simple	Simple	Simple	Single

CardHolder

Description: The information stored in this entity holds attributes that a cardholder would have when they create an account. These attributes include information such as address, first name, last name, etc.

Primary Key:

ID

Candidate Keys:

ID

Address

Strong Entity

Field to be Indexed:

ID

Attribute Name	ID	LName	Fname	Address	State	Phone	Email	Bdate	PrintID	LateFees	LibraryID	Electronic ID
Description	ID for internal data base	Card holder last name	Card holder first name	Card holder Address	Cardholder State	Card holder phone number	Card holder Email	Birth date	Current print rentals	Accrued late fees	Library member made card	Current electronic rentals
Domain/ Type	Integer	String	String	String	String	Integer	String	Integer	Integer	Integer	Integer	Integer
Value-Range	0-Max	Any	Any	Any	Any	0-Max	Any	0-Max	0-Max	0-Max	0-Max	0-Max
Default Value	None	None	None	None	None	None	None	None	None	None	None	None
Null Allowed	No	No	No	No	No	No	No	No	No	Yes	No	No
Unique	Yes	No	No	No	No	No	Yes	No	Yes	No	Yes	Yes
Single/ Multivalue	Single	Single	Single	Single	Single	Single	Single	Single	Multivalue	Single	Single	Multivalue
Simple/ Composite	Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple

1.2.2 Relationship Set Descriptions

Employee Works for Library

Description: An employee is assigned to work at a library. Employees can be swapped between libraries, but we do not keep employment dates here since this is a superclass that just stores common characteristics.

Entity Sets Involved: Employee, Library

Mapping Cardinality: M...M

Descriptive Fields: None

Participation Constraints: Employee to Library is total participation. An employee must be assigned to one library. Library to employee is total participation. A library must be assigned to one employee.

Library hosts Study Rooms

Description: A library can host study rooms. Study rooms can not be swapped to different libraries, but study rooms can be removed from a library.

Entity Sets Involved: Library, StudyRooms

Mapping Cardinality: M….N

Descriptive Fields: None

Participation Constraints: Study rooms to library is total participation. A study room must be assigned to one library.

CardHolder requests StudyRooms

Description: A member with a card can request a study room. A member can be assigned to many study rooms, and many study rooms can be assigned to many members.

Entity Sets Involved: CardHolder, StudyRooms

Mapping Cardinality: M….M

Descriptive Fields: StartDateAndTime, EndDateAndTime

Participation Constraints: CardHolder to StudyRooms is total participation. A cardholder must be assigned to an available room. However, a study room can be assigned to multiple cardholders.

Employee orders Materials

Description: The head librarian orders materials that are brand new to the library. Many head librarians can order many materials at any time in any library.

Entity Sets Involved: Employee, Materials

Descriptive Fields: None

Participant Restraints: Employee to Materials is total participation. The head librarian is the only entity that can order books, and books are necessary for the library to keep up to date. Materials to Employee is total participation. Books can only be ordered by the Employee.

Materials Rented to CardHolder

Description: A material that is either print or electronic can be checked out or returned by a cardholder if they choose to rent it. Many Materials can be checked out by many cardholders.

Entity Sets Involved: Materials, CardHolder

Mapping Cardinality: M….M

Descriptive Fields: RentStart, RentEnd, CopyID

Participation Restraints: Materials to Cardholder is total participation. Materials can only be checked out and returned by a cardholder. Cardholder to Materials is total participation. A cardholder can get any amount of materials.

Materials to Electronic

Description: Electronic is a subclass of Materials, this relationship allows us to store more specific information in the electronic subclass.

Entity Sets Involved: Materials, Electronic

Mapping Cardinality: M….M

Descriptive Fields: None

Participation Restraints: Materials to Electronic is total participation. Materials contains electronic. Electronic to Materials is total participation. Electronics are a type of Material.

Materials to Print

Description: Print is a subclass of Materials, this relationship allows us to store more specific information in the print subclass.

Entity Sets Involved: Materials, Print

Mapping Cardinality: M….M

Descriptive Fields: None

Participation Restraints: Materials to Print is total participation. Materials contains print. Print to Materials is total participation. Print are a type of Material.

Employee contains WorkLog

Description: In order to keep track of working hours the entity Employee contains the WorkLog entity. This will allow hours, dates, and time worked to be tracked for both paid and non paid employees. Many Employees can log many WorkLogs.

Entity Sets Involved: Employee, WorkLog

Mapping Cardinality: M….M

Descriptive Fields: None

Participation Restraints: Employee to WorkLog is total participation. A library technician must fill out a work log. WorkLog to Employee is total participation. A work log is necessary for an employee to work.

1.2.3 Related Entity Set Descriptions

Employee to HeadLibrarian and LibraryTechnician

Superclass: Employee

Subclasses: HeadLibrarian, LibraryTechnician

Description: Employee is a superclass for both Headlibrarian and LibraryTechnician due to both positions sharing many common attributes. This allows us to organize employees easier, and separate what makes each employee unique from each other.

Entity Sets Involved: Employee, HeadLibrarian, LibraryTechnicians

Materials to Print and Electronic

Superclass: Materials

Subclasses: Print, Electronic

Description: Materials is a superclass for both Print and Electronic. This allows the consolidation of common attributes between both materials found in the library. It also allows us to search for items easier within the database when they are requested by a cardholder.

Entity Sets Involved: Materials, Print, Electronic

1.2.4 E-R Diagram

Phase 2

2.1 ER Model and Relational Model

The conceptual modeling of a database is an extremely crucial phase in designing a successful application. Usually, database application refers to a specific database and the programs that are associated with implementing both database queries and updates. These programs will usually be

associated with a graphical user interface that allows users easier access to the application through easy to read menus and forms. The database application will require one of these application programs to be designed and implemented. The design and testing of an application program will not be part of database design, but software engineering.

2.1.1 Description of ER Model and Relation Modeling

Ted Codd of IBM Research introduced the relational data model in 1970, which attracted immediate attention due to its mathematical foundation as well as its simplicity. Mathematical relations are used as a basic building block, which are similar to a table of values, and uses first-order predicate logic as its basis in set theory. The SQL language is used as the standard for relational database management systems. Relational calculus is used through relational algebra, and these two languages are associated with the relational model. SQL uses relational calculus as the basis for the SQL language, and the relational algebra is used in the internals of many databases for processing queries and optimization.

2.1.2 Comparison of ER Model and Relational Model

An ER model is a summary of all information which can be found within a database as well as arranged within a database, it is represented as a high level graphical model. Its usefulness comes from being able to be understood by a large range of users and personnel. Some features that are part of the entity-relational model are attributes, relations, entities, and the cardinality found between entities.More accurately, the entity-relational model comes into fruition during and after gathering all requirements and analyzing all data from database analysts.

The relational model seeks to aid the use of relational algebra, tuple relational calculus, and domain relational calculus by creating a visual summary of how the database is structured.The relational model may describe some real world application and relationships, but primarily it is used to describe a specific relational database more than a conceptual model with its associated query languages.

2.2 Conversion from ER Model to Relational Model

The ER model may describe requirements for data, but the model can not be used to write a database directly. Instead, the model must be mapped to an implementation model similar to the relational model.

Step One: A relation is created for all simple attributes for every regular entity type. If multiple keys are found and identified during the conceptual design, the information used to describe the

attributes is kept in order to specify secondary keys. Entity relations are the relations which are created from the mapping of entity types.

Step Two: A weak entity is handled when a relation is created that includes all simple and simple component attributes of the weak entity. The owner entity gives its primary key in the form of both a foreign key and primary key. The partial key is also included as part of the primary key.

Step Three: Binary 1:1 Relationships

A. A method used when at least one or two relations that have total participation would be including the primary key of one of the two relations.

B. Another method is merging both relations into a single relation if both relations have total participation in the relationship.

C. Two relations may be cross-referenced by creating a third relation. The third relation contains foreign keys that refer back to the primary keys of both participating relations in the relation.

Step Four: Binary M:N or N:1 relationships can be mapped by two different methods.

A. First, include the primary key of the 1-side relation as a foreign key inside of the N-side relation.

B. The second method is to first check if the level of participation of the N-side relation is low, in this case two relations may instead be cross-referenced.

Step Five: Binary M:N relationships are always required to be cross-reference. Usually, foreign keys are utilized to delete as well as update operations.

Step Six: When an attribute is multivalued it is required to be implemented in a new relation. The foreign key will reference the primary key of the relation that represents the entity or relationship that originally contained the attribute.

Step Seven: Ternary and N relationships are also required to be cross-referenced.The relation participating in the relationship will use its primary key to implement a foreign key in the cross-reference table.

Step Eight: Specialization, or generalization, can be condensed into four methods.

A. When a relation for a super class and sub class is created the primary key of the super class will be implemented as a foreign key within the corresponding sub class.

B. When a relation is created for a sub class the attributes must be moved to the corresponding super class relation, which will produce a full set of attributes for specialization.

C. When a single with a single type attribute is created every super class and sub class will be merged into that single relation, but numerous nulls may be the result of this.

D. A single relation with multiple type attributes must be created. The types being implemented will be split into type attributes that will be Boolean. Due to this, some sub classes may overlap.

Step Nine: Union types, or categories, can be mapped in two different ways.

A. A surrogate key is created that consists of any category attributes and new key attributes, this surrogate key is placed in the newly created relation.

B. In the case that all super classes share a primary key then the creation of a surrogate key is needed. Instead, use the common primary key in this method.

Step Ten: If a recursive relationship is present then a new foreign key in the relation that referenced the primary key is necessary.

2.2.1 Constraints

When a constraint is found within the ER model it will also be implemented as constraints in the relational database. As an example, In order to guarantee that every attribute in every tuple is the correct data type there must be domain constraints implemented on those attributes in the relation.When a null constraint is used it guarantees that when a given attribute is present in all tuples in a given relation that the value will never be null.

In order to guarantee that every primary key within a certain relationship is unique an entity integrity constraint can be used. This method also ensures that within all tuples found in the relation that an attribute is unique. In order to guarantee that for any instantiated foreign key attribute in which a corresponding primary key attribute exists within the given relation a referential integrity constraint must be used.

The DBMS implements constraints that are schema based or explicit, this is generally done through settings in the database schema. However, business rules is another type of constraint that exists which can be explained.

Business rules is a constraint that can be implemented in many ways, some of which are triggers and check restraints. When a set of conditions is used before committing changes to any relation, this is called a check restraint. A check restraint is performed automatically in order to check the conditions at all times. The other implementation is triggers, which is SQL code that runs after an operation like an update, delete, or insert statement which run automatically as well.

3.1 Relation Schema

Library

Library(**ID**, Address, State, Zipcode, City, Phone)

1. **ID** - Domain: Integer
    A. Default Value: (Max ID + 1)
    B. Primary Key (implies Unique and NOT NULL)

2. Address - Domain: Varchar(50)
    A. NOT NULL
3. State - Domain: Varchar(3)
    A. NOT NULL
4. Zipcode - Domain: Integer(20)
    A. NOT NULL
5. City - Domain: Varchar(20)
    A. NOT NULL
6. Phone - Domain: Integer(10)
    A. NOT NULL

No other constraints

Candidate Keys

1. **ID**(primary key)

Employee

Employee(**ID**, Lname, Fname, SSN, Address, State, Phone, Email, Bdate, Payrate, Position, LibraryID)

1. **ID** - Domain: Integer
    A. Default Value: (Max ID + 1)
    B. Primary Key (implies Unique and NOT NULL)
2. Lname - Domain: Varchar(50)

    A. NOT NULL

3. Fname - Domain: Varchar(50)
    A. NOT NULL
4. SSN - Domain: Integer (9)
    A. NOT NULL
5. Address - Domain: Varchar(50)
    A. NOT NULL
6. State - Domain: Varchar(2)
    A. NOT NULL
7. Phone - Domain: Integer(10)
    A. NOT NULL
8. Email - Domain: Varchar(25)
    A. NOT NULL
9. Bdate - Domain: Date(If time is used, default to midnight)
    A. NOT NULL
10. Payrate - Domain: Integer(5)
    A. NOT NULL
11. Position - Domain: Varchar(15)
    A. NOT NULL
12. LibraryID - Domain: Integer
    A. Foreign key to Library.ID
    B. NOT NULL

No other Constraints

Candidate Keys

1. **ID** (Primary Key)

WorkLog

WorkLog(**Date**, TimeStart, Hours, EmployeeID)

1. Date - Domain: Date

    A. Default Value: Date
    B. Primary Key (implies Unique and NOT NULL)
2. TimeStart - Domain: Time
    A. NOT NULL
3. Hours - Domain: Time
    A. NOT NULL
4. EmployeeID - Domain: Integer
    A. Foreign key to Employee.ID
    B. NOT NULL

No other Constraints

Candidate Keys

1. **Date** (primary key)

Print

Print(**ID**, Publisher, IssueDate, NumCopy, ISBN, LibraryID, Title, Year, Genre, Author)

1. **ID** - Domain: Integer

   A. Default Value: (Max ID + 1)
   B. Primary Key(implies Unique and NOT NULL)
2. Publisher - Domain: Varchar(50)
   A. NOT NULL
3. IssueDate - Domain: Date
4. NumCopy - Domain: Integer(10)
   A. NOT NULL
5. Author - Domain: Varchar(50)
   A. NOT NULL
6. ISBN - Domain: Integer(13)
   A. NOT NULL
7. Title - Domain: Varchar(50)
   A. NOT NULL
8. Year - Domain: Date
9. Genre - Domain: Varchar(50)
   A. NOT NULL
10. LibraryID - Domain: Integer
   A. Foreign key to Library.ID
   B. NOT NULL

No other Constraints

Candidate Keys

1. ID (Primary Key)

Electronic

Electronic(ID,Production, Visual, NumCopy, Title, Year, Genre, LibraryID)

1. **ID** - Domain: Integer

    A. Default Value: (Max ID + 1)
    B. Primary Key (implies Unique and NOT NULL)
2. SELT - Domain: Integer(13)
3. Production - Domain: Varchar(50)
4. Visual - Domain: Boolean(2)
    A. NOT NULL
5. NumCopy - Domain: Integer(10)
    A. NOT NULL
6. Title - Domain: Varchar(50)
    A. NOT NULL
7. Year - Domain: Date
8. Genre - Domain: Varchar(50)
    A. NOT NULL
9. LibraryID - Domain: Integer
    A. Foreign key to Library.ID
    B. NOT NULL

No other Constraints

Candidate Keys

1. **ID** (primary key)

StudyRooms

StudyRooms(**ID**, Capacity, Purpose, Location, LibraryID)

1. **ID** - Domain: Integer

   A. Default Value: (Max ID + 1)
   B. Primary Key (implies Unique and NOT NULL
2. Capacity - Domain: Integer(10)
   A. NOT NULL
3. Purpose - Domain: Varchar(20)
   A. NOT NULL
4. Location - Domain: Varchar(10)
   A. NOT NULL
5. LibraryID - Domain: Integer
   A. Foreign Key to library.ID
   B. NOT NULL

No other constraints

Candidate Keys

1. **ID** (Primary Key)

CardHolder

CardHolder(**ID**, Lname, Fname, Address,State, Phone, Email, Bdate, LateFees, LibraryID, PrintID, ElectronicID)

1. **ID** - Domain: Integer
    A. Default Value: (Max ID + 1)
    B. Primary Key (implies Unique and NOT NULL)
2. Lname - Domain: Varchar(20)
    A. NOT NULL
3. Fname - Domain: Varchar(20)
    A. NOT NULL
4. Address - Domain: Varchar(50)
    A. NOT NULL
5. State - Domain: Varchar(2)
    A. NOT NULL
6. Phone - Domain: Integer(10)
    A. NOT NULL
7. Email - Domain: Varchar(50)
    A. NOT NULL
8. Bdate - Domain: Date
    A. NOT NULL
9. LateFees - Domain: Integer(5)
10. LibraryID - Domain: Integer
    A. Foreign Key to library.ID
    B. NOT NULL
11. PrintID - Domain: Integer
    A. Foreign Key to Print.ID
    B. NOT NULL
12. ElectronicID - Domain: Integer
    A. Foreign Key to Electronic.ID
    B. NOT NULL

No other constraints

Candidate Keys

1. **ID** (Primary Key)

3.2 Relation Instances

Library

Library(**ID**, Address, State, Zipcode, City, Phone)

ID	Address	State	Zipcode	City	Phone
1	123 What St.	CA	93309	Bakersfield	661-432-1235
2	341 Free St.	CA	93309	Bakersfield	661-123-5678
3	122 Sum St.	CA	93309	Bakersfield	661-852-7662
4	512 Tree St.	CA	93311	Delano	662-312-1244
5	154 Little Ave.	OH	71893	Cleveland	765-123-5433
6	617 Eighth Dr.	OH	71893	Cleveland	765-432-5442
7	986 Drive Dr.	OH	12345	Cincinnati	769-098-0986
8	861 Counter St.	NV	98000	Reno	657-231-8956
9	565 Parry Ave.	NV	98000	Reno	657-546-5435
10	1236 Jump St.	NV	72673	Las Vegas	651-342-5478

Employee

Employee(**ID**, Lname, Fname, SSN, Address, State, Phone, Email, Bdate, Payrate, Position, LibraryID)

ID	Lname	Fname	SSN	Address	State	Phone	Email	Bdate	Payrate	Position	Library ID
1	Harriet	William	575 23 674 1	312 What St.	CA	661-231 -1233	WHarriet@gmail.com	12/3/82	12.00	Library Technician	1
2	Jones	Tanya	656 32 657 2	321 No St.	CA	661-132 -2315	TJones@gmail.com	4/13/98	0.00	Volunteer	1
3	Fisher	Tom	563 21 763 3	42 Common Ave.	OH	676-123-5572	hamsterlover13@hotmail.com	6/23/87	40.00	Head Librarian	2
4	James	Ken	742 15 531 3	643 Hello St.	OH	651-235-1561	J3dIwannab3@yahoo.com	8/27/96	12.0	Library Technician	2
5	Ren	Chris	543 12 689 8	23 Bye Dr.	NV	543-213-6554	CRen@gmail.com	7/18/97	12.00	Library Technician	3
6	Ibarra	Albert	132 48 653 1	561 Scott Ave.	NV	763-569-2411	ahbangmahknee87@aol.com	6/4/93	0.00	Volunteer	4
7	Tanori	Daniel	512 57 908 9	651 Runner St.	CA	987-751-5573	DTanori@gmail.com	9/12/76	40.00	Head Librarian	5
8	Tuck	James	564	512	OH	855-213-	JTuck	8/3/8	12.00	Library	6

			12 671 3	Juniper Dr.		6587	@gmail.com	5		Technician	
9	Sam	Neil	124 23 155 7	123 Capree St.	NV	151-124-4512	NSam@gmail.com	5/2/87	40.00	Head Librarian	6
10	Alric	Alvin	613 12 412 1	45 Constant Ave.	NV	512-123-5411	AA@gmail.com	6/12/98	0.00	Volunteer	8

WorkLog

WorkLog(**Date**, TimeStart, Hours, EmployeeID)

Date	TimeStart	Hours	EmployeeID
10/8/17	4:00 PM	3:00	1
10/1/17	1:00 PM	6:00	1
9/2/17	11:00 AM	8:00	2
8/23/16	12:00 PM	2:00	3
2/27/17	1:00 PM	5:00	4
4/2/17	10:00 AM	8:00	4
3/1/16	4:00 PM	2:00	5
4/6/17	6:00 PM	2:00	6
5/1/17	10:00 AM	5:00	8
8/6/17	2:00 PM	4:00	8

Print

Print(**ID**, ISBN, Publisher, IssueDate, NumCopy, Author, LibraryID, Title, Year, Genre)

ID	ISBN	Publisher	Issue Date	Num Copy	Author	LibraryID	Title	Year	Genre
001	2345121153	White Wolf	1986	2	Steven King	1	IT	1986	Horror
002	1231234513	Giant	1949	2	George Orwell	1	1984	1949	Science Fiction
003	5664272312	Adventure Inc	1985	1	Steven King	1	Carrie	1985	Horror
004	5664272312	Adventure Inc	1985	1	Steven King	2	Carrie	1985	Horror
005	5555121328	White Wolf	1932	1	Aldous Huxley	3	Brave New World	1932	Science Fiction
006	54837458123	Hello Books	1978	1	Steven King	1	The Stand	1978	Science Fiction
007	54837458123	Hello Books	1978	1	Steven King	5	The Stand	1978	Science Fiction
008	54837458123	Hello Books	1978	2	Steven King	5	The Stand	1978	Science Fiction
009	4124125563	Giant	1990	2	Michael Crichton	5	Jurassic Park	1990	Science Fiction
010	678562132	Goodbye Books	2011	1	Ernest Cline	7	Ready Player One	2011	Science Fiction
011	983248234	Yellow Inc	1952	1	Ralph Ellison	1	The Invisibl	1952	Horror

							e Man		
012	983248234	Yellow Inc	1952	1	Ralph Ellison	8	The Invisible Man	1952	Horror
013	123182848	Edison	1954	1	J.R.R. Tolkien	9	The Lord of The Rings	1954	Fantasy
014	1294737818	Hello Books	1925	1	F. Scott Fitzgerald	4	The Great Gatsby	1925	Fiction
015	1294737818	Hello Books	1925	2	F. Scott Fitzgerald	4	The Great Gatsby	1925	Fiction

Electronic

Electronic(**ID**, Production, Visual, NumCopy, LibraryID, Title, Year, Genre)

ID	Production	Visual	Num Copy	LibraryID	Title	Year	Genre
1	White Wolf	Y	3	1	One Flew Over the Cuckoo's Nest	1975	Fiction
2	Glass House	N	5	4	Jurassic Park	1993	Science Fiction
3	Image	Y	5	3	The Lord of The Rings	2001	Fantasy
4	IDD	N	6	7	IT	2017	Horro
5	Warner Bros	Y	3	1	The Great Gatsby	2013	Fiction
6	Fox	N	3	2	Full Metal Jacket	1987	Fiction
7	CBS	N	4	8	Vertigo	1958	Fiction
8	Giant	Y	5	9	2001: A Space Odyssey	1968	Science Fiction
9	Image	N	4	10	Fight Club	1999	Fiction
10	Edison	Y	3	6	Aladdin	1982	Animated

StudyRooms

StudyRooms(**ID**, Capacity, Purpose, Location, LibraryID)

ID	Capacity	Purpose	Location	LibraryID
1	4	Study	203	1
2	4	Study	401	1
3	6	Visual Room	101	2
4	4	Study	103	3
5	8	Large Group	207	3
6	2	Study	210	4
7	8	Large Group	304	7
8	6	Visual Room	211	6
9	4	Study	309	8
10	4	Study	106	4

CardHolder

CardHolder(**ID**, Lname, Fname, Address, State, Phone, Email, Bdate, LateFees, LibraryID, MaterialID, ElectronicID)

D	Lname	Fname	Address	State	Phone	Email	Bdate	Late Fees	Library ID	Material ID	Electronic ID
	Valle	Jeff	415 Free St.	CA	142 124 5123	JValle@gmail.com	10/4/67	0.00	1	001,002,003	1,2,4
2	Davidson	Arthur	123 Nowhere Dr.	CA	121 765 6654	ADavidson@gmail.com	2/1/78	5.00	3	004	1,2
3	Cameron	James	7645 Camp St.	OH	987 125 6546	JCameron@gmail.com	4/2/87	3.00	2	005	5,6,7
4	Hitchcock	Alfred	987 High Dr.	OH	655 135 4766	AHitchcock@gmail.com	6/7/90	0.00	6	007,001	3,4
5	Bradley	David	988 Scott St.	CA	765 124 3675	DBradley @gmail.com	7/9/98	0.00	5	008	3,7,8
6	Lynch	David	3234 Empire Ave.	NV	409 709 2544	DLynch@gmail.com	6/6/98	1.00	8	004	8,9,10
7	Hardwick	Chris	1234 Chen St.	NV	109 534 6534	CHardwick@gmail.com	5/12/88	0.00	8	010,003	6,3,4
8	Wesker	Albert	764	CA	469	AWesker@gm	7/6/87	0.00	4	005	1,2,5

			Sunrise Ave.		543 1651	ail.com					
9	Redfield	Claire	731 Farm St.	OH	545 7641 123	CRedfield@gmail.com	8/7/98	0.00	9	007	4,6,8,9
10	Acosta	Derrick	654 Mega Ave.	CA	347 876 6567	DAcosta@gmail.com	7/9/87	0.00	4	001,003,006	1,2,5,6

4.1 Sample Queries

The purpose of the sample queries we created was to see if everything found within the relational database schema could be connected to each other with real world applicable scenarios.

Library (LID, LAdress, LState, LZipcode, LCity, LPhone)

Employee (EID, EFName, ELName, ESSN, EBdate, EPosition, EAdress, EState, EPhone, EEmail, EPayrate, EPosition,ElibraryID)
WorkLog (WDate, WTimeStart, WHours, WEmployee ID)
Print (PID, PISBN, PPublisher, PTitle, PYear, PGenre, PIssueDate, PNumCopy, PAuthor, PLibraryID)
Electronic (EID, ElProduction, ElVisual, ElNumCopy, ElTitle, ElYear, ElGenre, ElLibraryID)
StudyRoom (SID, SCapacity, SPurpose, SLocation, SLibraryID)
CardHolder (CID, CLName, CFname, CAddress, CState, CPhone, CEmail, CBdate, CLateFees, CPrintID, CElectronicID, CLibraryID)

1. Name of books with different editions

$$p1.PAuthor = p2.PAuthor \wedge p1.PPublisher = p2.PPublisher \wedge$$
$$\sigma(printxprint)¿$$
$$\pi_{p.PTitle}¿$$
$$p1.PYear = p2.PYear \wedge p1.PISBN \neq p2.PISBN \quad ¿¿$$

$$\{P \vee Print(p) \wedge (\exists P_2)(Print(P_2) \wedge P.ISBN \neq P_2.ISBN)\}$$

{<ID, ISBN, Publisher, Title, Year, Genre, IssueDate, NumCopy, Author, LibraryID> | <{P| Print(p) ,

,

ISBN, ,

,

¿,¿,¿,¿¿}

2. Name of cardholders with more than 3 checked-out items of the same genre

$$\sigma_{(cardholder \times print \times print \times print \times print)} \; (p1.PGenre = \text{genre} \wedge$$

$$\pi_{c.CLName, c.CFName} \quad p2.PGenre = \text{genre} \wedge p3.PGenre = \text{genre} \wedge \quad p4.PGenre = \text{genre} \wedge$$

$$p1.PGenre = \text{genre} \wedge c.CPrintID = p1.PID \wedge c.CPrintID = p2.PID$$

$$\wedge c.CPrintID = p3.PID \wedge c.CPrintID = p4.PID$$

$$Print(p) \wedge Print(p_2) \wedge Print(p_3) \wedge P.ID = C.PrintID$$

$$CardHolder(c) \wedge (\exists p)(\exists p_2)(\exists p_2)(\exists p_3)$$

$$\{ \wedge P_2.ID = C.PrintID \wedge P_3.ID = C.PrintID \wedge P.Genre = P_2.Genre \wedge P_2.Genre = P_3.Genre \}$$

{<LName, FName, Address, State, Phone, Email, Bdate, LateFees, ID, PrintID, ELectronicID,

LibraryID> | CardHolder(, , ,) $\wedge (\exists genre)(\exists genre_2)(\exists genre_3)$

, , , , , ,

$PrintID, , Genre, , \wedge$

$print$

$PrintID, , Genre, , \wedge$

$print$

, , ,

$print(PrintID, , Genre, ,)\}$

3. Name of cardholders with late fees greater than $100

$$\pi_{c.CLName, c.CFName}\left(\sigma(cardholders)(c.CLateFees > \$100)\right)$$

$$\{\forall CardHolder(c) \wedge (C.LateFees > \$100)\}$$

{<ID, LName, FName, Address, State, Phone, Email, Bdate, LateFees, PrintID, ElectronicID,

,
,
LibraryID>|CardHolder(,
¿, LName, FName, ¿, ¿, $100, ¿, ¿}

4. ID of printed books with electronic versions

$$\pi_{e.ElTitle}(\sigma(Print \times Electronic)(p.PTitle = e.ElTitle \wedge p.PGenre = e.ElGenre))$$

$$\{P \mid Print(p) \wedge (\exists e)(Electronics(e) \wedge p.Title = e.Title)\}$$

ID,
Print¿
{<ID, ISBN, Publisher, Title, Year, Genre, IssueDate, NumCopy, Author, LibraryID>¿¿

,
,
,
,
Electronic(¿, ¿, title, ¿)}
title, ¿, ¿¿ ∧ (∃title$_2$)¿

5. ID of study rooms with highest capacity on each library in the city

$$\pi_{l.LName}\left(\sigma\left(\text{StudyRoom x StudyRoom x Library}\right)(s1.SCapacity > s2.SCapacity \wedge s.SLibraryID \neq s2.SLibraryID \wedge l1.LCity = l2.LCity)\right)$$

$$Library(l) \wedge l.State = \text{CA} \wedge l.City = \text{Los Angeles} \wedge l.ID = S.LibraryID \{S \mid StudyRoom(s) \wedge (\exists s_2)(StudyRoom(s) \wedge s.ID \neq s_2.ID \wedge s.Capacity > s_2.Capacity \wedge S.Location \neq s_2.Location)\}$$

$$\{<Capacity, Purpose, Location, ID, Library> \mid StudyRoom(Capacity, Location, ID, LibraryID) \wedge (\exists state)(\exists city)(Library, CA, Los Angeles, ID = LibraryID) \wedge (\exists ID_2)(\exists location_2)(\exists capacity_2)(capacity > capacity_2, location = location_2, ID \neq ID_2,)\}$$

6. Head librarians that have worked at more than one library in the last year

$$\sigma(Employee \times Employee \times Library \times Library)$$
$$e1.EID = e2.EID \wedge$$
$$e1.Position = \text{Head Librarian} \wedge l1.LID = l2.LID \wedge l1.LAdress = l2.LAdress$$
$$\pi_{e.EFName, e.ELName}$$

$$library(l) \wedge l.ID = e.LibraryID \wedge e.Position = \text{Head Librarian} \wedge$$
$$\{e \mid Employee(e) \wedge (\exists l)$$
$$WorkLog(w_2)$$
$$WorkLog(w) \wedge w.Date = 1/1/2007 \wedge (\exists w_2)$$
$$library(l_2) l.ID \neq l_2.ID \wedge (\exists w)$$
$$(\exists_2)$$
$$\wedge w_2.DateL = 1/1/18\}$$

$$\{<ID, LName, FName, SSN, Address, State, Phone, Email, Bdate, PayRate, Position, LibraryID>$$
$$,$$
$$,$$
$$,$$
$$ID = LibraryID,$$
$$Library$$
$$Employee(ID, , , , HeadLibrarian, LibraryID) \wedge (\exists ID_2)$$
$$,$$
$$,$$
$$,$$
$$, , \wedge (\exists ID_3)(Date \geq 1/1/2017, ,) \wedge (\exists date)(Date) = 1/1/2018, , \}$$

7. ID of all fiction books published in year 2011

$$\pi_{p.pTitle}(\sigma(Print)(p.PYear = 2011 \wedge p.PGenre = \text{Fiction}))$$

$$\{P \mid Print(p) \wedge (\exists p_2)(Print(p_2) \wedge p.PID \neq p_2.PID \wedge P.PGenre = \text{Fiction} \wedge p.Pyear = 2011)\}$$

$$\{ <ID, ISBN, Publisher, Title, Year, Genre, IssueDate, NumCopy, Author, LibraryID> ¿$$

$$,$$
$$,$$
$$,$$
$$ID \neq ID2, ¿, 2011,$$
$$Print ¿$$
$$ID, ¿, 2001, Fiction, ¿, ¿ \wedge (\exists p_{id2})(\exists_{genre2})(\exists_{years}) ¿$$
$$Print ¿$$

$$,$$
$$Fiction, ¿, ¿ ¿\}$$

8. Find the member who has read all “Stephen King” books

$$p1.PAuthor = \text{Stephen King} \wedge p1.PISBN \neq$$
$$\sigma(Print \times Print \times cardholder) ¿$$
$$\pi_{c.CFName \wedge c.FLName} ¿$$
$$p2.PISBN \wedge c.CPrintID = p1.PID ¿ ¿$$

$$CardHolder(c) \wedge (\forall p)(Print(p) \wedge p.Author = \text{Stephen King} \rightarrow c.PrintID = p.ID)\}$$
$$\{ C ¿$$

$$\{ <LName, FName, Address, State, Phone, Email, Bdate, LateFees, ID, PrintID, ElectronicID,$$

$$,$$
$$,$$
$$¿$$
$$,$$
$$,$$
$$PrintID, ¿, ¿,$$
$$¿, ¿, ¿, ID, PrintID, ¿, (\forall author) ¿$$
$$LibraryID ¿ > ¿ Cardholder ¿$$

¿, Stephen King, $p2.PISBN$¿→$c.PrintID=p.ID$¿}

9. Name of Employees with greater than 40 hours logged in one week

$$e.ID=w.WEmployeeID \wedge w.WHours=40 \wedge$$
$$\sigma(Employee \times WorkLog)$$¿
$$\pi_{e.EFName, e.ELName}$$¿
$$w.WTimeStart \leq 7\ days$$¿¿

$$\{E \mid Employee(e) \wedge (\exists w)(WorkLog(w) \wedge w.Hours>40 \wedge w.Start>7\ days)\}$$

{ <ID, LName, FName, SSN, Address, State, Phone, Email, Bdate, PayRate, Position, LibraryID>¿
,
,
,
,
¿, <7 days, >40 hours, ¿}
ID, ¿, ¿, ¿, ¿ ∧ (∃worklog) ¿
¿Employee¿

10. Name of employees and cardholders with a birthday in October

$$\sigma(Employee \times CardHolder)(e.EBDate=10/*/* \wedge c.CBdate=10/*/*)$$
$$\pi_{e.Name}$$¿

$$\{\mid Employee(e) \wedge CardHolder(c) \wedge (\exists e)(e.Bdate=10/*/*) \wedge (\exists c)(c.Bdate=10/*/*)\}$$

{ <ID, LName, FName, SSN, Address, State, Phone, Email, Bdate, PayRate, Position, LibraryID>¿
∧<LName, FName, Address, State, Phone, Email, Bdate, LateFees, ID, PrintID, ElectronicID>¿

,
,
,
,
,
,
¿,¿,¿/ ¿/ ¿,¿,¿}
Employee(¿,¿,¿, 1 0/ ¿/ ¿,¿) /*CardHolder* ¿

Phase 3

3.1 Normalization of Relations

3.1.1 Normalization

Normalization is the process of organizing the relations and attributes of the database to bypass as much repetitiveness in data as possible. This is done by simplifying the relations in smaller tables and adding a foreign key into the newly made table from the original primary table. This is the solution to bypass any deletion, insertion, or update errors in the dataset.

There are three primary levels of normalization used for a relational database, as well as one modified version of the last normalization method. The first normal form(1NF) advises against having a set of values, a tuple of values, or both as an attribute value for a single tuple. To begin, the first normal form is seen as a portion of the appropriate definition of relation in the relational model in which multivalued and composite attributes are not approved. These methods are used together in the

first normal form standard; no relations found within relations and no multivalued or composite attributes in any single relation.

The second normal form (2NF) is founded on the idea of full functional dependency. A functional dependency (X → Y) is seen as full functional dependency if removed of any element A from X means that the dependency is no longer valid; which for an element A □X, (X - {A}) does not functionally determine Y. A functional dependency (X → Y) is defined as a partial dependency if some element A □ X can be removed from X and the dependency would still hold do A □ X, (X - {A}) → Y. A relation schema B is 2NF if every non-primary attribute A in B is fully and functionally dependent on the primary key of B.

The third normal form (3NF) is based on the idea of a transitive dependency. Again, a functional dependency X → Y in a relation schema B is a transitive dependency if there also exists a set of elements Z in B that is both not a candidate key and not a subset of any key of B, and both X → Y and Z → Y hold. According to the original definition presented by Codd; a relation schema B is in 3NF if it meets the requirements of 2NF and not a single non-prime attribute found in B is transitively dependent on the primary key.

The last form is Boyce-Codd Normal Form (BCNF); and is somewhat alike to 3NF in that it is seen as a more rigid version of 3NF. A relation in 3NF can also be in BCNF, but a relation in BCNF must also be in 3NF. A relation schema B is in BCNF if a non-trivial functional dependency X → A holds in B, then X is a superkey of B.

Update, insertion, and deletion anomalies are all existing in non-normalized databases. When these anomalies occur it can be left with complete, partial, or missing information about tuples from different relations.

3.1.2 Anomalies

When relations have not been normalized we can encounter a number of issues. These main issues are known as insertion, deletion, and modification anomalies.

Insertion Anomalies

When relation schema are poorly defined an insertion anomaly is common. This is usually present when inserting a new table that needs NULL values to accommodate the fact that no information has been inserted into the new table. This can lead to integrity constraint issues and the only workarounds are to insert NULL values into the attributes or to normalize the relation.

Deletion Anomalies

The main issue with deletion anomalies is connected to second insertion anomaly. If an entity in a table is deleted; that could potentially be the last amount of information in that table. Thus, the information in that table would now be completely lost.

Modification Anomalies

An update anomaly occurs when a poorly designed schema attempts to update a value in one tuple and causes an inconsistency across many other tuples in a relation. For example, if an attribute is changed to a different value we can run into a problem where tuples that contain data related to that attribute now have incorrect information unless they are updated as well. If this is not done correctly

then we are left with massive data inconsistencies from this single action.

3.1.3 Relation Normalization

Our database relations seek to follow all the rules set by Boyce-Codd forms. For the First Normal form all of the data in our relations are atomic values, and every attribute is split enough to make it indivisible. In relation to the Second Normal Form all our primary keys contain only a single value, so the test for Second Normal form does not apply in this situation. In the case of the Third Normal Form our foreign keys do not interact with other keys in the same schema object; so we did not need to split the relations into another relation.

3.2 Postgres SQL

Postgres is an object-relational database management system with an emphasis put on extensibility. The primary function so Postgres are to store data securely and return that data in response to requests from other software applications. In addition to this Postgres also supports a large part of the SQL standard and offers features including:

- Complex SQL queries
- SQL Sub-selects

- Foreign Keys
- Trigger
- Views
- Transactions
- Streaming Replication
- Hot Standby

3.3 Schema objects for Postgres SQL DBMS

3.3.1 Tables

Tables are the basic unit of data in a database. Data is stored in rows, and these rows contain the attributes of the relational schema. In a table; columns have names(such as E_ID, FName, LName), a specific datatype (Integer, Timestamp, Varchar), and a width for the allotted data. Once data is inserted

into the table it can then be updated, queried, or deleted using the SQL language. Tables can also contain virtual columns that derive values on request through user specified functions.

Syntax:

```
CREATE TABLE table_Name
(
        columnName1 dataType,
        columnName2 dataType,
        ...
        columnN datatype,
        PRIMARY KEY(one or more columns)
);
```

3.3.2 Views

A view is a logical, or virtual, table that is based on an already stored query that gets data from already existing base tables. Views aid in the construction of GUI's; due to the application pulling information from views rather than base tables.

Syntax:

```
CREATE VIEW viewName AS
SELECT ColumnNames1, ColumnNames2, ….
FROM tableName
WHERE [condition];
```

3.3.3 Functions

Functions are a set of SQL statements that when used together; perform a specific task on the database. They can take parameters and use those parameters to encapsulate a set of operations.

Syntax:

```
CREATE FUNCTION function_name
RETURNS return_datatype AS $variable_name$
    DECLARE
        declaration;
        [ . . . ]
    BEGIN
        < function_body >
        [ . . . ]
        RETURN { variable_name | value }
    END; LANGUAGE plpgsql;
```

3.3.4 Triggers

Triggers are procedures that are executed implicitly whenever an update, insert, or delete command is requested and executed on the database.

Syntax:

```
CREATE TRIGGER trigger_name
ON table_name
[
        --Trigger logic goes here--
];
```

3.3.5 Indexes

An index is an object that contains an entry for each value that every indexed column contains. An index allows for fast and direct access to rows found within the database, and this allows queries to perform much more efficiently.

Syntax:

```
CREATE INDEX  indexName
ON tableName (columnName);
```

3.3.6 Packages

A package is a grouping of SQL objects and procedures that provide an interface to more complicated SQL functionality. They encapsulate data and functions similarly to classes in object-oriented programming. Packages contain a block that defines the public interface to the public that is called a spec block. They also contain black that fully defines the hidden and abstracted code within procedures called the “body” block. Packages allow complex procedures to be easily hidden from a front-end application developer.

3.4 Relation Schema and Data

Library:

```
library=# \d+ library
                                                  Table "public.library"
 Column  |         Type          |                    Modifiers                     | Storage  | Stats target | Description
---------+-----------------------+--------------------------------------------------+----------+--------------+-------------
 lid     | integer               | not null default nextval('seqlibid'::regclass) | plain    |              |
 address | character varying(40) | not null                                         | extended |              |
 state   | character varying(2)  | not null                                         | extended |              |
 zip     | integer               | not null                                         | plain    |              |
 city    | character varying(25) | not null                                         | extended |              |
 phone   | bigint                | not null                                         | plain    |              |
Indexes:
    "library_pkey" PRIMARY KEY, btree (lid)
    "idx_library_id" btree (lid)
Referenced by:
    TABLE "cardholder" CONSTRAINT "cardholder_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
    TABLE "electronic" CONSTRAINT "electronic_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
    TABLE "employee" CONSTRAINT "employee_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
    TABLE "print" CONSTRAINT "print_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
    TABLE "studyrooms" CONSTRAINT "studyrooms_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
    TABLE "worksfor" CONSTRAINT "worksfor_libid_fkey" FOREIGN KEY (libid) REFERENCES library(lid)

library=#
```

```
library=# select * from library
library-# ;
 lid |     address      | state |  zip  |    city     |   phone
-----+------------------+-------+-------+-------------+------------
   1 | 123 What St.     | CA    | 93309 | Bakersfield | 6614321235
   2 | 341 Free St.     | CA    | 93309 | Bakersfield | 6611235678
   3 | 122 Sum St.      | CA    | 93309 | Bakersfield | 6618527662
   4 | 512 Tree St.     | CA    | 93311 | Delano      | 6623121244
   5 | 154 Little Ave.  | OH    | 71893 | Cleveland   | 7651235433
   6 | 617 Eighth Dr.   | OH    | 71893 | Cleveland   | 7654325442
   7 | 986 Drive Dr.    | OH    | 12345 | Cleveland   | 7690980986
   8 | 861 Counter St.  | NV    | 98000 | Reno        | 6572318956
   9 | 565 Parry Ave.   | NV    | 98000 | Reno        | 6575465435
  10 | 1236 Jump St.    | NV    | 72673 | Las Vegas   | 6513425478
(10 rows)

library=#
```

Employee:

```
library=# \d+ employee
                                                  Table "public.employee"
  Column  |         Type          |                    Modifiers                    | Storage  | Stats target | Description
----------+-----------------------+-------------------------------------------------+----------+--------------+-------------
 eid      | integer               | not null default nextval('seqempid'::regclass) | plain    |              |
 lname    | character varying(30) | not null                                        | extended |              |
 fname    | character varying(20) | not null                                        | extended |              |
 ssn      | bigint                | not null                                        | plain    |              |
 address  | character varying(40) | not null                                        | extended |              |
 state    | character varying(2)  | not null                                        | extended |              |
 phone    | bigint                | not null                                        | plain    |              |
 email    | character varying(30) | not null                                        | extended |              |
 bdate    | date                  | not null                                        | plain    |              |
 payrate  | double precision      | not null                                        | plain    |              |
 position | character varying(20) | not null                                        | extended |              |
 lid      | integer               |                                                 | plain    |              |
Indexes:
    "employee_pkey" PRIMARY KEY, btree (eid)
    "employee_ssn_key" UNIQUE CONSTRAINT, btree (ssn)
    "idx_employee_id" btree (eid)
    "idx_employee_ssn" btree (ssn)
Foreign-key constraints:
    "employee_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
Referenced by:
    TABLE "worklog" CONSTRAINT "worklog_eid_fkey" FOREIGN KEY (eid) REFERENCES employee(eid)
    TABLE "worksfor" CONSTRAINT "worksfor_eid_fkey" FOREIGN KEY (eid) REFERENCES employee(eid)

library=#
```

```
library=# select * from employee
library-# ;
 eid |  lname   |   fname    |    ssn    |      address      | state |   phone    |           email            |   bdate    | payrate |      position      | lid
-----+----------+------------+-----------+-------------------+-------+------------+----------------------------+------------+---------+--------------------+-----
   1 | Harriet  | William    | 575236741 | 312 What St.      | CA    | 6612311233 | WHarriet@gmail.com         | 1982-12-03 |      12 | Library Technician |   1
   2 | Jones    | Tanya      | 656326572 | 321 No St,        | CA    | 6611322315 | TJones@gmail.com           | 1998-04-13 |       0 | Volunteer          |   1
   3 | Fisher   | Tom        | 563217633 | 42 Common Ave.    | OH    | 6761235572 | hamsterlover13@hotmail.com | 1987-06-23 |      40 | Head Librarian     |   2
   4 | James    | Ken        | 742155313 | 643 Hello St.     | OH    | 6512351561 | J3dIwannab3@yahoo.com      | 1996-08-27 |    12.5 | Library Technician |   2
   5 | Ren      | Chris      | 543126898 | 23 Bye Dr.        | NV    | 5432136554 | CRen@gmail.com             | 1997-07-18 |      12 | Library Technician |   3
   6 | Ibarra   | Albert     | 132486531 | 561 Scott Ave.    | NV    | 7635692411 | ahbangmahknee87@aol.com    | 1993-06-04 |       0 | Volunteer          |   4
   7 | Tanori   | Daniel     | 512579089 | 651 Runner st.    | CA    | 9877515573 | DTanori@gmail.com          | 1976-09-12 |      40 | Head Librarian     |   5
   8 | Tuck     | James      | 564126713 | 512 Juniper Dr.   | OH    | 8552136587 | JTUCK@gmail.com            | 1985-08-03 |      12 | Library Technician |   6
   9 | Sam      | Neil       | 124231557 | 123 Capree St.    | NV    | 1511244512 | NSamgmail.com              | 1987-05-02 |      40 | Head Librarian     |   6
  10 | Alric    | Alvin      | 613124121 | 45 Constant Ave.  | NV    | 5121235411 | AA@gmail.com               | 1998-06-12 |       0 | Volunteer          |   8
  11 | Eric     | Eric       | 613765121 | 454 What St.      | CA    | 5127865411 | EE@gmail.com               | 1998-09-22 |      40 | Head Librarian     |   8
  12 | Evans    | Jerry      | 676424121 | 453 No St.        | CA    | 5165425411 | JEvans@gmail.com           | 1990-07-15 |      40 | Head Librarian     |   9
  13 | Seinfeld | Jenny      | 677854121 | 984 Constant Ave. | NV    | 1623425411 | JSeinfeld@gmail.com        | 1997-12-15 |      40 | Head Librarian     |  10
  14 | Gates    | Sally      | 677897641 | 924 Juniper Dr.   | OH    | 4512325411 | SGates@gmail.com           | 1993-01-16 |      40 | Head Librarian     |   1
  15 | Head     | Kenny      | 677878965 | 851 Runner St.    | CA    | 1623401931 | KHead@gmail.com            | 1992-01-03 |      40 | Head Librarian     |   3
  16 | Rojas    | Karen      | 647354121 | 163 What St.      | CA    | 1652315411 | KRojas@gmail.com           | 1993-02-13 |      40 | Head Librarian     |   7
  17 | Lemon    | Lilly      | 786524121 | 765 Constant Ave. | NV    | 1667725411 | LLemon@gmail.com           | 1992-10-20 |      12 | Library Technician |   3
  18 | Lara     | Ramon      | 786546721 | 896 Common Ave.   | OH    | 1672125411 | RLara@gmail.com            | 1991-09-02 |      12 | Library Technician |  10
  19 | Hurtado  | Christopher | 778654121 | 654 Runner St.    | CA    | 5432725411 | CHurtado@gmail.com         | 1994-04-21 |      12 | Library Technician |   9
  20 | Murillo  | Christian  | 786424121 | 982 Juniper Dr.   | OH    | 1667727865 | CMurillo@gmail.com         | 1996-03-18 |      12 | Library Technician |   5
(20 rows)

library=#
```

WorkLog:

```
library=# \d+ worklog
                                                Table "public.worklog"
  Column   |         Type          |                    Modifiers                    | Storage  | Stats target | Description
-----------+-----------------------+-------------------------------------------------+----------+--------------+-------------
 wlid      | integer               | not null default nextval('seqwlid'::regclass) | plain    |              |
 date      | date                  | not null                                        | plain    |              |
 timestart | character varying(4)  | not null                                        | extended |              |
 hours     | double precision      | not null                                        | plain    |              |
 eid       | integer               |                                                 | plain    |              |
Indexes:
    "worklog_pkey" PRIMARY KEY, btree (wlid)
    "idx_worklog_id" btree (wlid)
Foreign-key constraints:
    "worklog_eid_fkey" FOREIGN KEY (eid) REFERENCES employee(eid)

library=#
```

```
library=# select * from worklog;
 wlid |    date    | timestart | hours | eid
------+------------+-----------+-------+-----
    1 | 2017-10-08 | 1600      |     3 |   1
    2 | 2017-10-01 | 1300      |     6 |   1
    3 | 2017-09-02 | 1100      |     8 |   2
    4 | 2016-08-23 | 1200      |     2 |   3
    5 | 2017-02-27 | 1300      |     5 |   4
    6 | 2017-04-02 | 1000      |     8 |   4
    7 | 2016-03-01 | 1600      |     2 |   5
    8 | 2017-04-06 | 1800      |     2 |   6
    9 | 2017-05-01 | 1000      |     5 |   7
   10 | 2017-08-06 | 1400      |     4 |   8
   11 | 2017-07-03 | 1200      |     5 |   9
   12 | 2017-09-12 | 1000      |     4 |  10
   13 | 2017-07-02 | 1300      |     5 |  11
   14 | 2017-07-03 | 0900      |     3 |  12
   15 | 2017-10-03 | 1200      |     5 |  13
   16 | 2017-01-04 | 0900      |     6 |  14
   17 | 2017-03-02 | 1000      |     3 |  15
   18 | 2017-04-12 | 0900      |     7 |  16
   19 | 2017-09-14 | 1200      |     4 |  17
   20 | 2017-05-04 | 0800      |     8 |  18
   21 | 2017-07-16 | 0900      |     3 |  19
   22 | 2017-03-15 | 0900      |     4 |  20
   23 | 2017-10-03 | 1200      |     4 |   2
   24 | 2017-07-11 | 1000      |     5 |   3
   25 | 2017-06-20 | 1100      |     4 |   5
```

```
   25 | 2017-06-20 | 1100       |      4 |    5
   26 | 2017-08-22 | 0900       |      3 |    6
   27 | 2017-06-20 | 0800       |      6 |    7
   28 | 2017-07-25 | 1300       |      5 |    8
   29 | 2017-01-05 | 1100       |      3 |    9
   30 | 2017-03-18 | 1200       |      4 |   10
   31 | 2017-03-15 | 1300       |      5 |   11
   32 | 2017-05-02 | 1400       |      3 |   12
   33 | 2017-03-20 | 1000       |      5 |   13
   34 | 2017-06-17 | 1400       |      2 |   14
   35 | 2017-10-02 | 0800       |      6 |   15
   36 | 2017-07-10 | 1200       |      5 |   16
   37 | 2017-06-22 | 1200       |      4 |   17
   38 | 2017-07-24 | 1000       |      6 |   18
   39 | 2017-08-22 | 0800       |      4 |   19
   40 | 2017-10-04 | 1200       |      5 |   20
   41 | 2017-04-22 | 0900       |      6 |    1
   42 | 2017-05-17 | 1400       |      3 |    2
   43 | 2017-08-11 | 1200       |      4 |    3
   44 | 2017-03-11 | 0900       |      8 |    4
   45 | 2017-08-12 | 1000       |      3 |    5
(45 rows)

library=#
```

Print:

```
library=# \d+ print
                                Table "public.print"
  Column   |          Type          | Modifiers | Storage  | Stats target | Description
-----------+------------------------+-----------+----------+--------------+-------------
 printid   | integer                | not null  | plain    |              |
 isbn      | bigint                 | not null  | plain    |              |
 publisher | character varying(50)  | not null  | extended |              |
 issuedate | character varying(4)   | not null  | extended |              |
 pnumcopy  | integer                | not null  | plain    |              |
 author    | character varying(50)  | not null  | extended |              |
 lid       | integer                |           | plain    |              |
 title     | character varying(50)  | not null  | extended |              |
 year      | integer                | not null  | plain    |              |
 genre     | character varying(25)  | not null  | extended |              |
Indexes:
    "print_pkey" PRIMARY KEY, btree (printid)
    "idx_print_isbn" btree (isbn)
Foreign-key constraints:
    "print_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
Referenced by:
    TABLE "rentprint" CONSTRAINT "rentprint_printid_fkey" FOREIGN KEY (printid) REFERENCES print(printid)

library=#
```

```
 printid |    isbn     |   publisher    | issuedate | pnumcopy |       author        | lid |          title
         | year |      genre
---------+-------------+----------------+-----------+----------+---------------------+-----+-------------------------
-+------+-----------------
       1 |  2345121153 | White Wolf     | 1986      |        2 | Steven King         |   1 | IT
 | 1986 | Horror
       2 |  1231234513 | Giant          | 1949      |        2 | George Orwell       |   1 | 1984
 | 1949 | Science Fiction
       3 |  5664272312 | Adventure Inc. | 1985      |        1 | Steven King         |   1 | Carrie
 | 1985 | Horror
       4 |  5664272312 | Adventure Inc. | 1985      |        1 | Steven King         |   2 | Carrie
 | 1985 | Horror
       5 |  5555121328 | White Wolf     | 1932      |        1 | Aldous Huxley       |   3 | Brave New World
 | 1932 | Science Fiction
       6 | 54837458123 | Hello Books    | 1978      |        1 | Steven King         |   1 | The Stand
 | 1978 | Science Fiction
       7 | 54837458123 | Hello Books    | 1978      |        1 | Steven King         |   5 | The Stand
 | 1978 | Science Fiction
       8 | 54837458123 | Hello Books    | 1978      |        2 | Steven King         |   5 | The Stand
 | 1978 | Science Fiction
       9 |  4124125563 | Giant          | 1990      |        2 | Mechael Crichton    |   5 | Jurassic Park
 | 1990 | Science Fiction
      10 |   678562132 | Goodbye Books  | 2011      |        1 | Ernest Cline        |   7 | Ready Player One
 | 2011 | Science Fiction
      11 |   983248234 | Yellow Inc     | 1952      |        1 | Ralph Ellison       |   1 | The Invisible Man
 | 1952 | Horror
      12 |   983248234 | Yellow Inc     | 1952      |        1 | Ralph Ellison       |   8 | The Invisible Man
 | 1952 | Horror
      13 |   123182848 | Edison         | 1954      |        1 | J.R.R. Tolkien      |   9 | Lord of the Rings
 | 1954 | Fantasy
```

```
     14 |  1294737818 | Hello Books    | 1925      |        1 | F. Scott Fitzgerald |   4 | The Great Gatsby
 | 1925 | Fiction
     15 |  1294737818 | Hello Books    | 1925      |        2 | F. Scott Fitzgerald |   4 | The Great Gatsby
 | 1925 | Fiction
     16 |  1342337818 | White Wolf     | 1940      |        1 | James Skeltor       |   4 | The The Skeleton
 | 1940 | Horror
     17 |  8976737818 | Giant          | 1998      |        1 | Duponte James       |   6 | There Be Dragons
 | 1998 | Fantasy
     18 |  1299807818 | Edison         | 1990      |        1 | Scott Steiner       |   7 | All The Math
 | 1990 | Biography
     19 |  1294737865 | Goodbye Books  | 2001      |        1 | Jack Skunk          |  10 | A Stinky Story
 | 1998 | Non-Fiction
     20 |  1675737818 | Yellow Inc     | 1967      |        1 | Thomas James        |   4 | Box Boy
 | 1967 | Fantasy
     21 |  1294897518 | White Wolf     | 1987      |        1 | John Sacks          |   9 | The Potato Bag
 | 1990 | Non-Fiction
     22 |  7654237818 | Giant          | 2008      |        1 | Pete Passer         |   8 | Big Pumpkin
 | 2008 | Horror
     23 |  1294787650 | Adventure Inc  | 2005      |        1 | Harold Ramis        |   5 | Buster
 | 2005 | Fiction
     24 |  1289637818 | Hello Books    | 2004      |        1 | Dave Buster         |   3 | The Big Bomb
 | 2004 | Science Fiction
     25 |  1289637819 | Hello Books    | 2014      |        1 | Dave Buster         |   1 | The Big Bomb 2nd Edition
 | 2014 | Science Fiction
(25 rows)
```

Electronic:

```
library=# \d+ electronic
                                                  Table "public.electronic"
    Column    |         Type          |                        Modifiers                        | Storage  | Stats target | Description
--------------+-----------------------+---------------------------------------------------------+----------+--------------+-------------
 electronicid | integer               | not null default nextval('seqelectronicid'::regclass) | plain    |              |
 production   | character varying(50) | not null                                                | extended |              |
 visual       | boolean               |                                                         | plain    |              |
 enumcopy     | integer               | not null                                                | plain    |              |
 lid          | integer               |                                                         | plain    |              |
 title        | character varying(50) | not null                                                | extended |              |
 year         | integer               | not null                                                | plain    |              |
 genre        | character varying(25) | not null                                                | extended |              |
Indexes:
    "electronic_pkey" PRIMARY KEY, btree (electronicid)
Foreign-key constraints:
    "electronic_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
Referenced by:
    TABLE "rentelectronic" CONSTRAINT "rentelectronic_electronicid_fkey" FOREIGN KEY (electronicid) REFERENCES electronic(electronicid)

library=#
```

```
library=# select * from electronic;
 electronicid | production  | visual | enumcopy | lid |             title              | year |      genre
--------------+-------------+--------+----------+-----+--------------------------------+------+-----------------
         5001 | White Wolf  | t      |        1 |   1 | One Flew Over the Cuckoos Nest | 1975 | Fiction
         5002 | Glass House | f      |        1 |   2 | Jurassic Park                  | 1993 | Science Fiction
         5003 | Image       | t      |        1 |   3 | The Lord of the Rings          | 2001 | Fantasy
         5004 | IDD         | f      |        1 |   4 | IT                             | 2017 | Horror
         5005 | Warner Bros | t      |        1 |   5 | The Great Batsby               | 2013 | Fiction
         5006 | Fox         | f      |        1 |   6 | Full Metal Jacket              | 1987 | Fiction
         5007 | CBS         | f      |        1 |   7 | Vertigo                        | 1958 | Fiction
         5008 | Giant       | t      |        1 |   8 | 2001: A Space odyssey          | 1968 | Science Fiction
         5009 | Image       | f      |        1 |   9 | Fight Club                     | 1999 | Fiction
         5010 | Edison      | t      |        1 |  10 | Aladdin                        | 1982 | Animated
         5011 | White Wolf  | f      |        1 |   1 | Boxed                          | 1988 | Horror
         5012 | White Wolf  | f      |        2 |   1 | Boxed                          | 1988 | Horror
         5013 | IDD         | t      |        1 |   2 | Boxer                          | 1999 | Fiction
         5014 | Glass House | t      |        1 |   3 | Raging Bull                    | 2004 | Non- Fiction
         5015 | Image       | t      |        1 |   4 | Song of Ice                    | 2002 | Fantasy
         5016 | Image       | f      |        1 |   5 | Highrise                       | 2001 | Science Fiction
         5017 | Edison      | t      |        1 |   6 | Space Odyssey                  | 2004 | Science Fiction
         5018 | Fox         | t      |        1 |   7 | The Odyssey                    | 2007 | Fiction
         5019 | Giant       | t      |        1 |   8 | Continue?                      | 2003 | Fantasy
         5020 | Warner Bros | t      |        1 |   9 | Summer                         | 2004 | Fiction
(20 rows)

library=#
```

CardHolder:

```
library=# \d+ cardholder
                                                  Table "public.cardholder"
  Column  |         Type          |                     Modifiers                      | Storage  | Stats target | Description
----------+-----------------------+----------------------------------------------------+----------+--------------+-------------
 chid     | integer               | not null default nextval('seqchid'::regclass) | plain    |              |
 lname    | character varying(30) | not null                                           | extended |              |
 fname    | character varying(15) | not null                                           | extended |              |
 address  | character varying(50) | not null                                           | extended |              |
 state    | character varying(2)  | not null                                           | extended |              |
 phone    | bigint                | not null                                           | plain    |              |
 email    | character varying(25) | not null                                           | extended |              |
 bdate    | date                  | not null                                           | plain    |              |
 latefees | double precision      | not null                                           | plain    |              |
 lid      | integer               |                                                    | plain    |              |
Indexes:
    "cardholder_pkey" PRIMARY KEY, btree (chid)
    "idx_cardholder_id" btree (chid)
    "idx_cardholder_lname" btree (lname)
Foreign-key constraints:
    "cardholder_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
Referenced by:
    TABLE "rentelectronic" CONSTRAINT "rentelectronic_chid_fkey" FOREIGN KEY (chid) REFERENCES cardholder(chid)
    TABLE "rentprint" CONSTRAINT "rentprint_chid_fkey" FOREIGN KEY (chid) REFERENCES cardholder(chid)
    TABLE "requests" CONSTRAINT "requests_chid_fkey" FOREIGN KEY (chid) REFERENCES cardholder(chid)

library=#
```

```
library=# select * from cardholder;
 chid |   lname   |  fname   |     address      | state |   phone    |        email         |   bdate    | latefees | lid
------+-----------+----------+------------------+-------+------------+----------------------+------------+----------+-----
    1 | Valle     | Jeff     | 415 Free St.     | CA    | 1421245123 | JValle@gmail.com     | 1967-10-04 |        0 |   1
    2 | Davidson  | Arthur   | 123 Nowhere Dr.  | CA    | 1217656654 | ADavidson@gmail.com  | 1978-02-01 |        5 |   2
    3 | Cameron   | James    | 7645 Camp St.    | OH    | 9871256546 | JCameron@gmail.com   | 1987-04-02 |        3 |   3
    4 | Hitchcock | Alfred   | 987 High Dr.     | OH    | 6551354766 | AHitchcock@gmail.com | 1990-06-07 |        0 |   4
    5 | Bradley   | David    | 988 Scott St.    | CA    | 7651243675 | DBradley@gmail.com   | 1998-07-09 |        0 |   5
    6 | Lynch     | David    | 3224 Empire Ave. | NV    | 4097092544 | DLynch@gmail.com     | 1998-06-06 |        1 |   6
    7 | Hardwick  | Chris    | 1234 Chen St.    | NV    | 1095346534 | CHardwick@gmail.com  | 1988-05-12 |        0 |   7
    8 | Wesker    | Albert   | 764 Sunrise Ave. | CA    | 4695431651 | AWesker@gmail.com    | 1987-07-06 |        0 |   8
    9 | Redfield  | Claire   | 731 Farm St.     | OH    | 5457641123 | CRedfield@gmail.com  | 1998-08-07 |      100 |   9
   10 | Acosta    | Derrick  | 654 Mega Ave.    | CA    | 3478766567 | DAcosta@gmail.com    | 1987-07-09 |        0 |  10
   11 | Burberry  | Jill     | 541 Scott St.    | CA    | 8976466567 | JBurberry@gmail.com  | 1989-02-12 |        5 |   1
   12 | Chucka    | Bill     | 542 Nowhere Dr.  | CA    | 4653766567 | BChucka@gmail.com    | 2001-05-03 |        0 |   2
   13 | Klein     | John     | 763 Camp St.     | OH    | 4521766567 | JKlein@gmail.com     | 1983-10-22 |        0 |   3
   14 | Stinson   | Daniel   | 431 High Dr.     | OH    | 3474356767 | DStinson@gmail.com   | 1992-11-15 |        5 |   4
   15 | Stein     | Alfred   | 772 Farm St.     | OH    | 3472315767 | AStein@gmail.com     | 1992-05-11 |        6 |   5
   16 | Scott     | Den      | 786 Empire Ave.  | NV    | 3476754367 | DScott@gmail.com     | 2002-03-15 |        2 |   6
   17 | Chin      | Jen      | 561 High St.     | OH    | 4567154316 | JChin@gmail.com      | 2001-10-15 |        0 |   7
   18 | Martinez  | Salvador | 873 Camp St.     | OH    | 3474378667 | SMartinez@gmail.com  | 1999-06-01 |        0 |   8
   19 | Hurtado   | Hector   | 980 Empire Ave.  | NV    | 3478438577 | HHurtado@gmail.com   | 1992-09-11 |        0 |   9
   20 | Ibarra    | Sarah    | 987 Chen St.     | NV    | 3478679547 | SIbarra@gmail.com    | 2001-08-14 |        0 |  10
(20 rows)

library=#
```

StudyRoom:

```
library=# \d+ studyrooms
                                                  Table "public.studyrooms"
  Column  |         Type          |                        Modifiers                        | Storage  | Stats target | Description
----------+-----------------------+---------------------------------------------------------+----------+--------------+-------------
 srid     | integer               | not null default nextval('seqsrid'::regclass) | plain    |              |
 capacity | integer               | not null                                                | plain    |              |
 purpose  | character varying(25) | not null                                                | extended |              |
 location | integer               | not null                                                | plain    |              |
 lid      | integer               |                                                         | plain    |              |
Indexes:
    "studyrooms_pkey" PRIMARY KEY, btree (srid)
    "idx_studyrooms_id" btree (srid)
Foreign-key constraints:
    "studyrooms_lid_fkey" FOREIGN KEY (lid) REFERENCES library(lid)
Referenced by:
    TABLE "requests" CONSTRAINT "requests_srid_fkey" FOREIGN KEY (srid) REFERENCES studyrooms(srid)

library=#
```

```
library=# select * from studyrooms ;
 srid | capacity |   purpose   | location | lid
------+----------+-------------+----------+-----
    1 |        4 | Study       |      203 |   1
    2 |        4 | Study       |      401 |   2
    3 |        6 | Visual Room |      101 |   3
    4 |        4 | Study       |      103 |   4
    5 |        8 | Large Group |      207 |   5
    6 |        2 | Study       |      210 |   6
    7 |        8 | Large Group |      304 |   7
    8 |        6 | Visual Room |      211 |   8
    9 |        4 | Study       |      309 |   9
   10 |        4 | Study       |      106 |  10
   11 |        5 | Study       |      201 |   1
   12 |        6 | Large Group |      221 |   2
   13 |        4 | Study       |      302 |   3
```

```
   14 |          4 | Visual Room |       226 |    4
   15 |          4 | Study       |       202 |    5
   16 |          4 | Study       |       105 |    6
   17 |          4 | Visual Room |       303 |    7
   18 |          5 | Study       |       313 |    8
   19 |          4 | Study       |       205 |    9
   20 |          5 | Study       |       323 |   10
   21 |          4 | Visual Room |       201 |    1
   22 |          6 | Large Group |       222 |    3
   23 |          4 | Study       |       336 |    5
   24 |          4 | Study       |       209 |    7
   25 |          4 | Visual Room |       505 |    9
(25 rows)

library=#
```

RentPrint:

```
library=# \d+ rentprint
                    Table "public.rentprint"
 Column  |  Type   | Modifiers | Storage | Stats target | Description
---------+---------+-----------+---------+--------------+-------------
 prstart | date    | not null  | plain   |              |
 prend   | date    | not null  | plain   |              |
 chid    | integer |           | plain   |              |
 printid | integer |           | plain   |              |
Foreign-key constraints:
    "rentprint_chid_fkey" FOREIGN KEY (chid) REFERENCES cardholder(chid)
    "rentprint_printid_fkey" FOREIGN KEY (printid) REFERENCES print(printid)

library=#
```

```
library=# select * from rentprint ;
  prstart   |   prend    | chid | printid
------------+------------+------+---------
 2017-02-14 | 2017-02-25 |    1 |       1
 2017-02-01 | 2017-02-08 |    1 |       3
 2017-02-03 | 2017-02-10 |    1 |       6
 2017-03-12 | 2017-03-19 |    1 |      11
 2017-05-11 | 2017-05-18 |    2 |       2
 2017-04-01 | 2017-04-08 |    2 |       5
 2017-01-11 | 2017-01-18 |    2 |       7
 2017-03-13 | 2017-03-20 |    3 |       4
 2017-02-10 | 2017-02-17 |    4 |       8
 2017-06-13 | 2017-06-20 |    5 |       9
 2017-04-22 | 2017-04-29 |    6 |      10
 2017-06-04 | 2017-06-11 |    7 |      12
 2017-03-18 | 2017-03-25 |    8 |      13
 2017-07-12 | 2017-07-19 |    9 |      14
 2017-06-14 | 2017-06-22 |   10 |      15
 2017-08-02 | 2017-08-02 |   11 |      16
 2017-03-11 | 2017-03-04 |   12 |      17
```

```
 2017-09-14 | 2017-09-21 |    13 |       18
 2017-06-10 | 2017-06-17 |    14 |       19
 2017-04-11 | 2017-04-18 |    15 |       20
 2017-06-10 | 2017-06-17 |    16 |       21
 2017-03-10 | 2017-03-17 |    17 |       22
 2017-06-02 | 2017-06-13 |    17 |       23
 2017-10-08 | 2017-10-15 |    18 |       24
 2017-10-01 | 2017-10-09 |    19 |        2
 2017-10-01 | 2017-10-09 |    19 |        6
 2017-10-11 | 2017-10-18 |    20 |       14
 2017-11-01 | 2017-11-09 |     4 |       15
 2017-09-04 | 2017-09-11 |     5 |        6
 2017-08-05 | 2017-08-12 |     6 |        8
 2017-10-19 | 2017-10-26 |     7 |        9
 2017-11-15 | 2017-11-22 |     8 |       10
(32 rows)

library=#
```

RentElectronic:

```
library=# \d+ rentelectronic
                    Table "public.rentelectronic"
    Column    |  Type   | Modifiers | Storage | Stats target | Description
--------------+---------+-----------+---------+--------------+-------------
 erstart      | date    | not null  | plain   |              |
 erend        | date    | not null  | plain   |              |
 chid         | integer |           | plain   |              |
 electronicid | integer |           | plain   |              |
Foreign-key constraints:
    "rentelectronic_chid_fkey" FOREIGN KEY (chid) REFERENCES cardholder(chid)
    "rentelectronic_electronicid_fkey" FOREIGN KEY (electronicid) REFERENCES electronic(electronicid)

library=#
```

```
library=# select * from rentelectronic;
  erstart    |   erend    | chid | electronicid
-------------+------------+------+--------------
 2017-11-15 | 2017-11-18 |    1 |         5001
 2017-12-10 | 2017-12-17 |    2 |         5002
 2017-10-15 | 2017-10-22 |    3 |         5003
 2017-01-02 | 2017-01-09 |    4 |         5004
 2017-01-10 | 2017-01-17 |    5 |         5005
 2017-01-20 | 2017-01-27 |    6 |         5006
 2017-02-04 | 2017-02-11 |    7 |         5007
 2017-02-12 | 2017-02-19 |    8 |         5008
 2017-02-21 | 2017-02-28 |    9 |         5009
 2017-03-02 | 2017-03-09 |   10 |         5010
 2017-03-10 | 2017-03-17 |   11 |         5011
 2017-03-22 | 2017-03-29 |   12 |         5012
 2017-04-04 | 2017-04-11 |   13 |         5013
 2017-04-14 | 2017-04-21 |   14 |         5014
```

```
 2017-04-22 | 2017-04-29 |   15 |          5015
 2017-05-02 | 2017-05-09 |   16 |          5016
 2017-05-12 | 2017-05-19 |   17 |          5017
 2017-05-21 | 2017-05-29 |   18 |          5018
 2017-06-05 | 2017-06-12 |   19 |          5019
 2017-07-04 | 2017-07-11 |   20 |          5020
 2017-07-15 | 2017-07-23 |    5 |          5002
 2017-08-02 | 2017-08-09 |    7 |          5004
 2017-08-11 | 2017-08-19 |    2 |          5006
 2017-08-21 | 2017-08-28 |   15 |          5008
 2017-09-02 | 2017-09-10 |    3 |          5010
 2017-09-11 | 2017-09-19 |    2 |          5001
 2017-09-20 | 2017-09-28 |    4 |          5003
 2017-10-01 | 2017-10-09 |    6 |          5005
 2017-10-11 | 2017-10-20 |    8 |          5007
 2017-10-21 | 2017-10-29 |   10 |          5009
(30 rows)

library=#
```

Works_for:

```
library=# \d+ worksfor
                    Table "public.worksfor"
 Column |  Type   | Modifiers | Storage | Stats target | Description
--------+---------+-----------+---------+--------------+-------------
 dstart | date    | not null  | plain   |              |
 dend   | date    |           | plain   |              |
 eid    | integer |           | plain   |              |
 libid  | integer |           | plain   |              |
Foreign-key constraints:
    "worksfor_eid_fkey" FOREIGN KEY (eid) REFERENCES employee(eid)
    "worksfor_libid_fkey" FOREIGN KEY (libid) REFERENCES library(lid)

library=#
```

```
library=# select * from worksfor;
   dstart   |    dend    | eid | libid
------------+------------+-----+-------
 2001-01-24 | 2016-12-11 |   1 |     1
 2017-01-12 |            |   2 |     1
 2016-02-01 |            |   3 |     2
 2013-02-11 |            |   4 |     2
 2011-11-14 |            |   5 |     3
 2003-12-11 |            |   6 |     4
 2011-10-18 | 2017-01-12 |   7 |     5
 2013-12-21 |            |   8 |     6
 2014-11-16 |            |   9 |     6
 2013-10-12 |            |  10 |     8
 2014-02-11 |            |  11 |     8
```

```
 2011-12-23 |              |   12 |      9
 2012-02-13 |              |   13 |     10
 2016-05-17 |              |   14 |      1
 2014-09-21 |              |   15 |      3
 2015-04-22 |              |   16 |      7
 2012-10-15 | 2014-11-29   |   17 |      3
 2012-12-23 |              |   18 |     10
 2013-03-11 |              |   19 |      9
 1999-03-18 |              |   20 |      5
 1999-03-21 | 2015-12-11   |   16 |     10
(21 rows)

library=#
```

Requests:

```
library=# \d+ requests
                                Table "public.requests"
 Column  |          Type          | Modifiers | Storage  | Stats target | Description
---------+------------------------+-----------+----------+--------------+-------------
 srsdate | date                   | not null  | plain    |              |
 srstime | character varying(4)   | not null  | extended |              |
 sredate | date                   | not null  | plain    |              |
 sretime | character varying(4)   | not null  | extended |              |
 srid    | integer                |           | plain    |              |
 chid    | integer                |           | plain    |              |
Foreign-key constraints:
    "requests_chid_fkey" FOREIGN KEY (chid) REFERENCES cardholder(chid)
    "requests_srid_fkey" FOREIGN KEY (srid) REFERENCES studyrooms(srid)

library=#
```

```
library=# select * from requests ;
  srsdate   | srstime |  sredate   | sretime | srid | chid
------------+---------+------------+---------+------+------
 2017-01-02 | 1200    | 2017-01-02 | 1400    |    1 |    1
 2017-01-05 | 1100    | 2017-01-05 | 1300    |    2 |    2
 2017-01-10 | 1300    | 2017-01-10 | 1400    |    3 |    3
 2017-01-15 | 0900    | 2017-01-15 | 1100    |    4 |    4
 2017-01-20 | 0800    | 2017-01-20 | 1200    |    5 |    5
 2017-01-25 | 1400    | 2017-01-25 | 1600    |    6 |    6
 2017-02-03 | 1600    | 2017-02-03 | 1700    |    7 |    7
 2017-02-06 | 1000    | 2017-02-06 | 1200    |    8 |    8
 2017-02-10 | 0900    | 2017-02-10 | 1000    |    9 |    9
 2017-02-16 | 0800    | 2017-02-16 | 1100    |   10 |   10
 2017-02-25 | 0900    | 2017-02-25 | 1100    |   11 |   11
 2017-03-02 | 1300    | 2017-03-02 | 1500    |   12 |   12
 2017-03-10 | 1000    | 2017-03-10 | 1200    |   13 |   13
 2017-03-15 | 1200    | 2017-03-15 | 1300    |   14 |   14
 2017-03-23 | 1000    | 2017-03-23 | 1200    |   15 |   15
```

```
 2017-04-05 | 1200     | 2017-04-05 | 1300     |    16 |    16
 2017-04-08 | 1200     | 2017-04-08 | 1400     |    17 |    17
 2017-04-14 | 1300     | 2017-04-14 | 1500     |    18 |    18
 2017-04-20 | 1500     | 2017-04-20 | 1700     |    19 |    19
 2017-04-27 | 1400     | 2017-04-27 | 1800     |    20 |    20
 2017-05-03 | 1400     | 2017-05-03 | 1600     |    21 |     1
 2017-05-10 | 1500     | 2017-05-10 | 1600     |    22 |     2
 2017-05-14 | 1200     | 2017-05-14 | 1400     |    23 |     3
 2017-05-22 | 0800     | 2017-05-22 | 1000     |    24 |     4
 2017-05-28 | 0900     | 2017-05-28 | 1100     |    25 |     5
 2017-06-05 | 0900     | 2017-06-05 | 1000     |     1 |     6
 2017-06-10 | 1100     | 2017-06-10 | 1300     |     2 |     7
 2017-06-14 | 1100     | 2017-06-14 | 1200     |     3 |     8
 2017-06-20 | 0800     | 2017-06-20 | 0900     |     4 |     9
 2017-06-27 | 1100     | 2017-06-27 | 1300     |     5 |    10
(30 rows)

library=#
```

3.5 Sample Queries with Postgres SQL

1. Name of books with different editions:

```
SELECT * FROM
        print p WHERE EXISTS
        (SELECT 1 from print WHERE author = p.author AND publisher = p.publisher AND NOT
        issuedate = p.issuedate);
```

Output:

```
library=# select * from print p where exists (select 1 from print where author = p.author and publisher = p.publisher and not issuedate = p.issuedate);
 printid |    isbn    |  publisher  | issuedate | pnumcopy |   author    | lid |          title          | year |     genre
---------+------------+-------------+-----------+----------+-------------+-----+-------------------------+------+---------------
      24 | 1289637818 | Hello Books | 2004      |        1 | Dave Buster |   3 | The Big Bomb            | 2004 | Science Fiction
      25 | 1289637819 | Hello Books | 2014      |        1 | Dave Buster |   1 | The Big Bomb 2nd Edition | 2014 | Science Fiction
(2 rows)

library=#
```

2. Name of cardholders with late fees greater equal to $100:

```
SELECT * FROM
        cardholder c WHERE c.latefees > 99;
```

```
postgres=# \c library
You are now connected to database "library" as user "postgres".
library=# select * from cardholder c where c.latefees > 99;
 chid |  lname   | fname  |   address    | state |   phone    |        email        |   bdate    | latefees | lid
------+----------+--------+--------------+-------+------------+---------------------+------------+----------+-----
    9 | Redfield | Claire | 731 Farm St. | OH    | 5457641123 | CRedfield@gmail.com | 1998-08-07 |      100 |   9
(1 row)
```

3. Name of materials that are both in print and video:

```
SELECT p.title FROM print p, electronic e WHERE p.title = e.title;
```

```
library=# select p.title from print p, electronic e where p.title = e.title;
     title
---------------
 Jurassic Park
 IT
(2 rows)
```

4. Which library has the largest study room:

SELECT l.lid, l.address, l.city, l.state, l.zip, s.capacity, s.location FROM library l, studyrooms s

WHERE s.lid = l.lid AND s.capacity = (SELECT MAX(capacity) FROM studyrooms);

```
library=# select l.lid, l.address, l.city, l.state, l.zip, s.capacity, s.location
from library l, studyrooms s where s.lid = l.lid and s.capacity = (select max(capacity) from studyrooms);
 lid |     address     |   city    | state |  zip  | capacity | location
-----+-----------------+-----------+-------+-------+----------+----------
   5 | 154 Little Ave. | Cleveland | OH    | 71893 |        8 |      207
   7 | 986 Drive Dr.   | Cleveland | OH    | 12345 |        8 |      304
(2 rows)
```

5. Name of all fiction books published in 2005:

SELECT * FROM print p WHERE p.year = '2005' AND p.genre = 'Fiction';

```
library=# select * from print p where p.year = '2005' and p.genre = 'Fiction';
 printid |    isbn    |   publisher   | issuedate | pnumcopy |    author    | lid | title  | year |  genre
---------+------------+---------------+-----------+----------+--------------+-----+--------+------+---------
      23 | 1294787650 | Adventure Inc | 2005      |        1 | Harold Ramis |   5 | Buster | 2005 | Fiction
(1 row)
```

6. Name of employee with 40 or more hours logged in a week:

SELECT e.lid, e.lname, e.fname, w.hours, w.date FROM employee e, worklog w

WHERE w.hours > 39 AND w.eid = e.eid;

```
library=# select e.lid, e.lname, e.fname, w.hours, w.date from employee e,
worklog w where w.hours > 39 and w.eid = e.eid;
 lid | lname | fname | hours |    date
-----+-------+-------+-------+------------
   3 | Ren   | Chris |    40 | 2017-08-19
(1 row)
```

7. Name of employees and cardholders with a birthday in October:

```
SELECT * FROM (SELECT lname, fname, bdate FROM employee ORDER BY lname) e
        CROSS JOIN (SELECT lname, fname, bdate FROM cardholder ORDER BY lname) c
        WHERE EXTRACT(MONTH FROM e.bdate) = 10
        AND EXTRACT(MONTH FROM c.bdate = 10;
```

```
library=# select * from (select lname, fname, bdate from employee order by lname) e cross join (select lname, fname,
bdate from cardholder order by lname) c where EXTRACT(MONTH FROM e.bdate) = 10 and EXTRACT(MONTH FROM c.bdate) = 10;
 lname | fname |   bdate    | lname | fname |   bdate
-------+-------+------------+-------+-------+------------
 Lemon | Lilly | 1992-10-20 | Chin  | Jen   | 2001-10-15
 Lemon | Lilly | 1992-10-20 | Klein | John  | 1983-10-22
 Lemon | Lilly | 1992-10-20 | Valle | Jeff  | 1967-10-04
(3 rows)
```

8. Show all cardholders and their late fees:

```
SELECT lname, fname, SUM (latefees) FROM cardholder GROUP BY lname, fname;
```

```
library=# select lname, fname, SUM (latefees) from cardholder GROUP BY lname, fname;
   lname   |  fname   | sum
-----------+----------+-----
 Lynch     | David    |   1
 Klein     | John     |   0
 Redfield  | Claire   | 100
 Stinson   | Daniel   |   5
 Chucka    | Bill     |   0
 Acosta    | Derrick  |   0
 Stein     | Alfred   |   6
 Wesker    | Albert   |   0
 Martinez  | Salvador |   0
 Valle     | Jeff     |   0
 Cameron   | James    |   3
 Bradley   | David    |   0
 Ibarra    | Sarah    |   0
 Davidson  | Arthur   |   5
 Hardwick  | Chris    |   0
 Scott     | Den      |   2
 Burberry  | Jill     |   5
 Hurtado   | Hector   |   0
 Hitchcock | Alfred   |   0
 Chin      | Jen      |   0
(20 rows)
```

9. Show all cardholders who have late fees:

SELECT lname, fname, SUM (latefees) FROM cardholder GROUP BY lname, fname
HAVING SUM (latefees) > 0;

```
library=# select lname, fname, SUM (latefees) from cardholder GROUP BY lname, fname HAVING SUM (latefees) > 0;
  lname   | fname  | sum
----------+--------+-----
 Lynch    | David  |   1
 Redfield | Claire | 100
 Stinson  | Daniel |   5
 Stein    | Alfred |   6
 Cameron  | James  |   3
 Davidson | Arthur |   5
 Scott    | Den    |   2
 Burberry | Jill   |   5
(8 rows)
```

10. Show possible amount to pay for employees without overtime in 1 month for month's budget:

SELECT lname, fname, payrate * 160 FROM employee;

```
library=# select lname, fname, payrate * 160 from employee;
  lname   |    fname    | ?column?
----------+-------------+----------
 Harriet  | William     |     1920
 Jones    | Tanya       |        0
 Fisher   | Tom         |     6400
 James    | Ken         |     2000
 Ren      | Chris       |     1920
 Ibarra   | Albert      |        0
 Tanori   | Daniel      |     6400
 Tuck     | James       |     1920
 Sam      | Neil        |     6400
 Alric    | Alvin       |        0
 Eric     | Eric        |     6400
 Evans    | Jerry       |     6400
 Seinfeld | Jenny       |     6400
 Gates    | Sally       |     6400
 Head     | Kenny       |     6400
 Rojas    | Karen       |     6400
 Lemon    | Lilly       |     1920
 Lara     | Ramon       |     1920
 Hurtado  | Christopher |     1920
 Murillo  | Christian   |     1920
(20 rows)
```

11. Show all current employees:

```
SELECT fname, lname from employee e FULL OUTER JOIN worksfor w ON e.eid = w.eid
        WHERE dend IS NULL;
```

```
library=# select fname, lname from employee e FULL OUTER JOIN worksfor w on e.eid = w.eid WHERE dend IS NULL;
    fname    |  lname
-------------+----------
 Tanya       | Jones
 Tom         | Fisher
 Ken         | James
 Chris       | Ren
 Albert      | Ibarra
 James       | Tuck
 Neil        | Sam
 Alvin       | Alric
 Eric        | Eric
 Jerry       | Evans
 Jenny       | Seinfeld
 Sally       | Gates
 Kenny       | Head
 Karen       | Rojas
 Ramon       | Lara
 Christopher | Hurtado
 Christian   | Murillo
(17 rows)
```

12. Show all employees paid more than Christian:

```
SELECT fname, lname, payrate FROM employee WHERE payrate > (SELECT MAX(payrate)
        FROM employee WHERE fname = 'Christian');
```

```
library=# SELECT fname, lname, payrate from employee WHERE payrate > (SELECT MAX(payrate)
FROM employee WHERE fname = 'Christian');
 fname  |  lname   | payrate
--------+----------+---------
 Tom    | Fisher   |      40
 Ken    | James    |    12.5
 Daniel | Tanori   |      40
 Neil   | Sam      |      40
 Eric   | Eric     |      40
 Jerry  | Evans    |      40
 Jenny  | Seinfeld |      40
 Sally  | Gates    |      40
 Kenny  | Head     |      40
 Karen  | Rojas    |      40
(10 rows)
```

13. Show all records of past employee records:

```
SELECT e.fname, e.lname, w.dstart, w.dend FROM employee e, worksfor w WHERE w.dend
        NOTNULL AND e.eid = w.eid;
```

```
library=# SELECT e.fname, e.lname, w.dstart, w.dend FROM employee e, worksfor w
WHERE w.dend NOTNULL and e.eid = w.eid;
  fname   |  lname   |   dstart   |    dend
----------+----------+------------+------------
 William | Harriet | 2001-01-24 | 2016-12-11
 Daniel  | Tanori  | 2011-10-18 | 2017-01-12
 Lilly   | Lemon   | 2012-10-15 | 2014-11-29
 Karen   | Rojas   | 1999-03-21 | 2015-12-11
(4 rows)
```

14. Show book(s) with the greatest number of copies available at one library:

```
SELECT lid, printid, title, author, genre, pnumcopy FROM print
        WHERE pnumcopy = (SELECT MAX(pnumcopy) FROM print);
```

```
library=# SELECT lid, printid, title, author, genre, pnumcopy FROM print WHERE pnumcopy
= (SELECT MAX(pnumcopy) FROM print);
 lid | printid |      title      |       author        |  genre  | pnumcopy
-----+---------+-----------------+---------------------+---------+----------
   4 |      26 | The Great Gatsby | F. Scott Fitzgerald | Fiction |        3
(1 row)
```

15. Show movies made before 1970 to be classified as a classic:

```
SELECT * FROM electronic e WHERE e.year <= 1970;
```

```
library=# SELECT * FROM electronic e WHERE e.year <= 1970;
 electronicid | production | visual | enumcopy | lid |         title         | year |      genre
--------------+------------+--------+----------+-----+-----------------------+------+-----------------
         5007 | CBS        | f      |        1 |   7 | Vertigo               | 1958 | Fiction
         5008 | Giant      | t      |        1 |   8 | 2001: A Space odyssey | 1968 | Science Fiction
(2 rows)
```

3.6 Data Loader

Large amounts of sample data must be loaded into the physical implementation of the database, so we must determine a method as to how that information is stored. Some of these methods include: manually writing SQL commands to insert information as well as using software applications that automatically create SQL statements. There are also insert scripts that transfer data from one file to another.

3.6.1 Using “insert” SQL statements

The simplest way to insert data into DBMS tables is to use the built in “insert” SQL statement provided by the system. There are two ways to insert data into a database:

1. Insert into <table_name> values (<expression_1>, ..<expression_n>)
2. Insert into <table_name> < select_query>

The first version of this statement lets the programmer decide specific values for each column in the table when inserting a record. The second version lets you query into the database and use the result of that query as the column values.

This is a tedious but thorough method to add information into a database, so much of our additions to our database were done with this method in mind in order to make sure data was specific to our queries. It is not very desirable for adding large amounts of data into a database, so the following methods are able to run SQL commands from files, which can then insert that data into the database. There are far faster methods than manually adding data which will be discussed.

3.6.2 Data Loader

Dr. Huaqing Wang developed a software application that uses a command interface in order to insert data into database tables from a separate text file. The user must specify many thing about the database before this is done, such as: the name of the database, the password, and the file to be used in the command line. The text file being used to add information to the database must also follow a specific format that specifies what data is being inserted and the specific tuple that the data should be inserted into. A useful feature is the ability to choose a character as a delimiter to separate columns through the command line. Once the text file is completed the data loader creates “insert into” SQL statements that fill in the exact areas specified by the user. This speeds up the process of inserting data into the database exponentially because the user does not have to use the “insert into” command for each separate piece of information.

3.6.3 Oracle SQL Developer

Another tool that allows for easier insertion of data into a database is Oracle SQL developer. This program can connect to the delphi database and insert into the tables with the necessary data. This is done through a graphical user interface where the user can view all the tables within the database and easily import data into them using CSV files. SQL developer is also capable of creating scripts if the user wishes to insert the data manually.

PHASE 4: Stored Subprograms, Packages, and Triggers

Phase 4 Introduction

In this phase, we will show why the use of PL/pgSQL will allow a proper implementation of our database using functions such as packages, triggers, and subprograms. PostgreSQL allows user-defined functions to be written in numerous languages, like SQL and C. The other languages used to write these functions are called procedural languages. However, the database has no built-in understanding of these languages used to create functions. To counteract this, special handlers that know these languages are used allow these functions to work. These handlers are used to either connnect postgres and the existing language, or for parsing, execution, and syntax analysis. PostgreSQL has four procedure languages available to developers, and they are: PL/Tcl, PL/Perl, PL/pgSQL, PL/Python.

4: Postgres PL/pgSQL

A loadable procedural language for PostgreSQL database system is PL/pgSQL. The desired goals for PL/pgSQL were to create a language that:

1. Adds contol structures to SQL
2. Can have the option to be defined as trusted by a server
3. Inherits all functions, operators, and types that are user-defined
4. Can create functions and trigger procedures
5. Can calculate complex computations
6. Offers ease of use to developers

A useful feature of PL/pgSQL is that functions it creates can be used anywhere that built-in functions are being used. PL/pgSQL is included by default for versions of PostgreSQL 9.0 and later.

4.1: Postgres PL/SQL advantages

Most relational databases use SQL as the query language, and so does PostgreSQL. This is because SQL is easy to learn and easily portable from database to database. However, each SQL statement is read individually by a database server that is using SQL code. This means that the database would have to wait for a query to be processed, read and understand the results, compute any necessary problems, and then send any necessary queries to the server. This would all be done for every query sent by the client application. This constant stream of data can cause network overhead and inter process communications if the user client is in a different area than the database server.

The advantage of PL/pgSQL is that it can group a block of queries inside the database in order to have one larger computations rather than many individual ones. This allows the simplicity of SQL while being able to have the power of a procedural language. This also comes with reduced client/server communication overhead. With PL/pgSQL extra trips between client and server are no longer necessary, multiple rounds of query parsing can be sidestepped, and the client does not need to communication intermediate results to the server. PL/pgSQL also allows the use of any operators and data types found within SQL.

Functional benefits of PL/SQL

1: PL/pgSQL functions can be declared to return the polymorphic types found in any element of any array.
2: Using the VARIADIC marker, PL/pgSQL functions can be declared to accept a variable amount of arguments.
3:Functions in PL/pgSQL can be created to return a “set” of any data typer that can be returned as a single instance.
4: A function can be created to return void if a return value is not found, in PL/pgSQL.
5: PL/pgSQL also allows for functions to be written to accept scalar or array data types as arguments supported by the server, and then return that same data type as a result.
6: PL/pgSQL will also accept or return any composite type that is called by name within a database

4.2: Structure of PL/pgSQL

PL/pgSQL is a language that is block-constructed and each written statement within a block is completed with the use of a semicolon. A semicolon is necessary for a block found within another block after the statement END. However, an END that terminates a function body does not need a semicolon to end its computation. All keywords in PL/pgSQL are case-insensitive and identifiers are converted to lowercase unless stated otherwise with double-quotation marks; this is how it is handled in ordinary SQL language.

Syntax:

```
[<<label>>]
[ DECLARE]
        declaration ]
BEGIN
        statement
        ...
END;
```

When a developer finds it necessary to identify a block for use in an EXIT statement, or to

qualify the names of the variables created in the block; then a label is used. A label must match the label at the block's beginning if it is given after END.

4.3: Stored Procedures in PL/pgSQL

Stored procedures and user-defined functions in PL/pgSQL are a set of procedural and SQL statements. Due to this, declarations, loops, and assignments can be stored on a database serv and can be called using a SQL interface.

Syntax:

```
CREATE OR REPLACE FUNCTION function_name(parameter TYPE, parameter TYPE,...)
        RETURN void AS $$
BEGIN
        INSERT INTO table_name VALUES(col1,col2,...)
END;
```

Example Statement:

```
CREATE OR REPLACE FUNCTION add_library(city VARCHAR(50), state VARCHAR(2)
        RETURN void AS $$
BEGIN
        INSERT INTO libraries VALUES (city, state);
END;
$$ LANGUAGE 'plpgsql';
```

Void can be specified as the return type if a stored procedure does not return any value, as shown in the example. Most stored procedures return one or more results, or do no return any value at all. After creating a stored procedure you can use the SELECT statement to call the add_library procedure: SELECT add_library(“Whittier”, 'CA');

You can also use the PERFORM add_library() on the statement above to call the add_library procedure from inside another function.

Returning a Single Result:

When needed, PL/pgSQL can also return a result set by specifying refcursor return type in the PostgreSQL procedure, as shown below:

Syntax:

```
CREATE OR REPLACE FUNCTION show_books() RETURNS refcursor AS $$
DECLARE
        ref refcursor;
BEGIN
        OPEN ref FOR SELECT book, author FROM books;
        RETURN ref;
END;
$$ LANGUAGE 'plpgsql'
```

4.4: Triggers in PL/pgSQL

Triggers can be used and defined in PL/pgSQL with the CREATE FUNCTION command, it must be declared as a function with no arguments and return trigger. If the function is expected to recieve arguments then it is specified in the CREATE TRIGGER portion of code. Trigger arguments are utilized through TG-ARGV[] which passes the arguments meant for the function.

TG_ARGV[]

An array of text is the data type for this function, these are recieved from the CREATE TRIGGER statement. The index for this function begins at 0, and invalid indexes result in null.

TG_OP

This trigger statement handles the data type 'text' which are a string of UPDATE, DELETE, TRUNCATE, OR INSERT that shows for which snippet of code the trigger was activated.

Triggers will be attached to a specified table or view and will execute the targeted function 'example_function" when specific requirements are met. When a PL/pgSQL function is called as a

trigger, the top-level block creates special variables automatically. NULL must be returned by a trigger function, or it must return a value from the record at the table that the trigger was activated for.

Syntax:

```
CREATE FUNCTION trigger_name() RETURNS trigger AS $example_name$
        BEGIN
                --Code here
                --IF and ELSE statements can be used here
                --Some SQL statements can be used here as well
END;
$example_name$ LANGUAGE 'plpgsql';
```

Trigger Procedure

```
CREATE TRIGGER trigger_example BEFORE INSERT OR UPDATE ON table_example
        FOR REACH ROW EXECUTE PROCEDURE trigger_example();
```

When the function has been created we must specify the trigger procedure, shown by the syntax example above. As said before, to create a trigger we must first create a function that builds the body of the trigger. No arguments can be used in the creation of the function, and we must specify that the return type is a trigger. This can be reiterated in a quick review of shown here:

Create Trigger:

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [
OR ... ] }
        ON table
        [ FROM referenced_table_name ]
        [ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY
DEFERRED } ]
        [ FOR [ EACH ] { ROW | STATEMENT } ]
        [ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name ( arguments )

where event can be one of:
        INSERT
        UPDATE [ OF column_name [, ... ] ]
        DELETE
        TRUNCATE
```

4.5: Packages in PL/pgSQL

Like Oracle, PL/pgSQL also features packages. A package is a schema object that groups logically related PL/pgSQL variables, constants, cursors, types, subprograms, and exceptions. A package is compiled and stored in the database where content can be shared between applications. Packages have specifications which declare the public items that can be referenced from outside the package. The package must have a body if cursors or subprograms are included. Queries for public cursors and code for public subprograms must be defined in the body. An example can be shon for creating a Package here:

Syntax:

```
CREATE OR REPLACE PACKAGE package_example [ CLAUSE ] DEFINER AS
        FUNCTION func_example1 (variables and their TYPE)
        FUNCTION func_example2 (variables and their TYPE)
        FUNCTION procedure_name (var_name TYPE);
        FUNCTION procedure_name2 (var_name TYPE);
        FUNCTION procedure_name3 (var_name TYPE);

        exception_example EXCEPTION;
        exception_example2 EXCEPTION;
END package_example;
/ LANGUAGE 'plpgsql';
```

AUTHID Clause

The AUTHID clause of the package specification determines whether the subprograms and cursors found within the package run with the privileges of their creator or caller, and whether the unqualified references to schema objects are resolved in the schema of the creator or caller.

ACCESSIBLE BY Clause

The ACCESSIBLE BY clause of the package lets you specify a white-list of PL/pgSQL units that can see the contents of the package.

There are some benefits to using packages in PL/pgSQL. Some of these uses include:

Logging – using this package a developer could leave all logging and debugging statements in code, and decide on logging output can be made at a later time. Combining logging with pipes for the output to be stored in tables separate from the active transaction.

Pipes - Pipes allows asynchronous communication between and number of producer/consumer database sessions on any number of pipes.

Batch Reporting – report packages that is more suitable to autonomous transactions than lagging, but useful information such as start time, duration, running totals, etc are given automatically at report end.

4.6: Stored Functions in PL/pgSQL

In section 4.3 it was mentioned that both stored and user-defined functions can be written as SQL statements. Using the SQL interface we can call user-defined functions that are stored on the database server, similar to procedures. The main difference between a user-defined function and a stored procedure is that a function usually returns a value, but a procedure does not.

Syntax:

```
CREATE OR REPLACE FUNCTION decrement(i INT) RETURNS INT AS $$
BEGIN
        RETURN i – 1;
END;
$$ LANGUAGE 'plpgsql';
```

To use the above function we can use the function call : SELECT decrement(1); This well then use the following function written, and return a value of -9. Both stored procedures and user-defined functions are created with the CREATE FUNCTION statement in PL/pgSQL. There are some differences between stored procedures and functions in database systems, such as:

	Procedure	Function
Use in expression		X
Return a value		X
Return values as OUT parameter	X	
Return a single result set	X	
Return multiple result sets	X	

4.7: Making Triggers, Procedures, and Placing them within a Package for a database

In this section, procedures, functions, and triggers will be created and executed for my own database. The rest of this section will detail each requirement of the database that is set for us. I created 3 procedures that allow us to add to the database, delete from the database, and calculate the average of fields N from a column in a table.

Insert Data Procedure Syntax & Result

The first procedure that was created for the database is called “add_cardholder()” that allows us to add a new cardholder to our library database. It completes this by running a Insert command with necessary values, which are chID, lname, fname, address, state, phone, email, bdate, latefees, and lID. In this example we add “Joey Frank” to the database.

```
postgres=# CREATE OR REPLACE FUNCTION new_cardholder(ChID integer, Lname varchar(30), Fname varchar(30), Address varchar(50), State varchar(2), Phone bigint, Email
 varchar(25), Bdate date, Latefees float, LID integer)
postgres-# RETURNS void AS $$
postgres$# BEGIN
postgres$# INSERT INTO cardholder VALUES( chID, lname, fname, address, state, phone, email, bdate, latefees, lID);
postgres$# END;
postgres$# $$ LANGUAGE 'plpgsql';
CREATE FUNCTION
postgres=#
```

```
postgres=# select new_cardholder('22', 'Frank', 'Joey', '231 Hello Ave.', 'CA', '3471132222', 'Joey@gmail.com', '2001-01-02', '20', '1');
 new_cardholder
----------------

(1 row)


postgres=# select * from cardholder;
 chid |   lname   |  fname   |     address      | state |   phone    |        email         |   bdate    | latefees | lid
------+-----------+----------+------------------+-------+------------+----------------------+------------+----------+-----
    1 | Valle     | Jeff     | 415 Free St.     | CA    | 1421245123 | JValle@gmail.com     | 1967-10-04 |        0 |   1
    2 | Davidson  | Arthur   | 123 Nowhere Dr.  | CA    | 1217656654 | ADavidson@gmail.com  | 1978-02-01 |        5 |   2
    3 | Cameron   | James    | 7645 Camp St.    | CA    | 9871256546 | JCameron@gmail.com   | 1987-04-02 |        3 |   3
    4 | Hitchcock | Alfred   | 987 High Dr.     | CA    | 6551354766 | AHitchcock@gmail.com | 1990-06-07 |        0 |   4
    5 | Bradley   | David    | 988 Scott St.    | CA    | 7651243675 | DBradley@gmail.com   | 1998-07-09 |        0 |   5
    6 | Lynch     | David    | 3224 Empire Ave. | CA    | 4097092544 | DLynch@gmail.com     | 1998-06-06 |        1 |   1
    7 | Hardwick  | Chris    | 1234 Chen St.    | CA    | 1095346534 | CHardwick@gmail.com  | 1988-05-12 |        0 |   2
    8 | Wesker    | Albert   | 764 Sunrise Ave. | CA    | 4695431651 | AWesker@gmail.com    | 1987-07-06 |        0 |   3
    9 | Redfield  | Claire   | 731 Farm St.     | CA    | 5457641123 | CRedfield@gmail.com  | 1998-08-07 |      100 |   4
   10 | Acosta    | Derrick  | 654 Mega Ave.    | CA    | 3478766567 | DAcosta@gmail.com    | 1987-07-09 |        0 |   5
   11 | Burberry  | Jill     | 541 Scott St.    | CA    | 8976466567 | JBurberry@gmail.com  | 1989-02-12 |        5 |   1
   12 | Chucka    | Bill     | 542 Nowhere Dr.  | CA    | 4653766567 | BChucka@gmail.com    | 2001-05-03 |        0 |   2
   13 | Klein     | John     | 763 Camp St.     | CA    | 4521766567 | JKlein@gmail.com     | 1983-10-22 |        0 |   3
   14 | Stinson   | Daniel   | 431 High Dr.     | CA    | 3474356767 | DStinson@gmail.com   | 1992-11-15 |        5 |   4
   15 | Stein     | Alfred   | 772 Farm St.     | CA    | 3472315767 | AStein@gmail.com     | 1992-05-11 |        6 |   5
   16 | Scott     | Den      | 786 Empire Ave.  | CA    | 3476754367 | DScott@gmail.com     | 2002-03-15 |        2 |   1
   17 | Chin      | Jen      | 561 High St.     | CA    | 4567154316 | JChin@gmail.com      | 2001-10-15 |        0 |   2
   18 | Martinez  | Salvador | 873 Camp St.     | CA    | 3474378667 | SMartinez@gmail.com  | 1999-06-01 |        0 |   3
   19 | Hurtado   | Hector   | 980 Empire Ave.  | CA    | 3478438577 | HHurtado@gmail.com   | 1992-09-11 |        0 |   4
   20 | Ibarra    | Sarah    | 987 Chen St.     | CA    | 3478679547 | SIbarra@gmail.com    | 2001-08-14 |        0 |   5
   21 | Frank     | Frank    | 231 Empire Ave.  | CA    | 3471112222 | Frank@gmail.com      | 2001-01-01 |       50 |   1
   22 | Frank     | Joey     | 231 Hello Ave.   | CA    | 3471132222 | Joey@gmail.com       | 2001-01-02 |       20 |   1
(22 rows)
```

Delete Existing Data Procedure & Result

The next procedure being shown will be able to delete an existing record in the cardholder table. It does this by running a Deletion command with proper syntax. The function will search through the table and find a cardholder that matches the entered chID that is placed in the function. If the record exists, our function will then delete that cardholder from that table. I will delete the "Joey Frank" cardholder that I added in the previous section.

```
postgres=# CREATE OR REPLACE FUNCTION delete_cardholder(drop integer)
postgres-# RETURNS void AS $$
postgres$# BEGIN
postgres$# DELETE FROM cardholder WHERE chID = drop;
postgres$# END;
postgres$# $$ LANGUAGE 'plpgsql';
CREATE FUNCTION
```

```
postgres=# SELECT delete_cardholder('22');
 delete_cardholder
-------------------

(1 row)


postgres=# SELECT * FROM cardholder;
 chid |   lname   |  fname   |     address      | state |   phone    |        email         |   bdate    | latefees | lid
------+-----------+----------+------------------+-------+------------+----------------------+------------+----------+-----
    1 | Valle     | Jeff     | 415 Free St.     | CA    | 1421245123 | JValle@gmail.com     | 1967-10-04 |        0 |   1
    2 | Davidson  | Arthur   | 123 Nowhere Dr.  | CA    | 1217656654 | ADavidson@gmail.com  | 1978-02-01 |        5 |   2
    3 | Cameron   | James    | 7645 Camp St.    | CA    | 9871256546 | JCameron@gmail.com   | 1987-04-02 |        3 |   3
    4 | Hitchcock | Alfred   | 987 High Dr.     | CA    | 6551354766 | AHitchcock@gmail.com | 1990-06-07 |        0 |   4
    5 | Bradley   | David    | 988 Scott St.    | CA    | 7651243675 | DBradley@gmail.com   | 1998-07-09 |        0 |   5
    6 | Lynch     | David    | 3224 Empire Ave. | CA    | 4097092544 | DLynch@gmail.com     | 1998-06-06 |        1 |   1
    7 | Hardwick  | Chris    | 1234 Chen St.    | CA    | 1095346534 | CHardwick@gmail.com  | 1988-05-12 |        0 |   2
    8 | Wesker    | Albert   | 764 Sunrise Ave. | CA    | 4695431651 | AWesker@gmail.com    | 1987-07-06 |        0 |   3
    9 | Redfield  | Claire   | 731 Farm St.     | CA    | 5457641123 | CRedfield@gmail.com  | 1998-08-07 |      100 |   4
   10 | Acosta    | Derrick  | 654 Mega Ave.    | CA    | 3478766567 | DAcosta@gmail.com    | 1987-07-09 |        0 |   5
   11 | Burberry  | Jill     | 541 Scott St.    | CA    | 8976466567 | JBurberry@gmail.com  | 1989-02-12 |        5 |   1
   12 | Chucka    | Bill     | 542 Nowhere Dr.  | CA    | 4653766567 | BChucka@gmail.com    | 2001-05-03 |        0 |   2
   13 | Klein     | John     | 763 Camp St.     | CA    | 4521766567 | JKlein@gmail.com     | 1983-10-22 |        0 |   3
   14 | Stinson   | Daniel   | 431 High Dr.     | CA    | 3474356767 | DStinson@gmail.com   | 1992-11-15 |        5 |   4
   15 | Stein     | Alfred   | 772 Farm St.     | CA    | 3472315767 | AStein@gmail.com     | 1992-05-11 |        6 |   5
   16 | Scott     | Den      | 786 Empire Ave.  | CA    | 3476754367 | DScott@gmail.com     | 2002-03-15 |        2 |   1
   17 | Chin      | Jen      | 561 High St.     | CA    | 4567154316 | JChin@gmail.com      | 2001-10-15 |        0 |   2
   18 | Martinez  | Salvador | 873 Camp St.     | CA    | 3474378667 | SMartinez@gmail.com  | 1999-06-01 |        0 |   3
   19 | Hurtado   | Hector   | 980 Empire Ave.  | CA    | 3478438577 | HHurtado@gmail.com   | 1992-09-11 |        0 |   4
   20 | Ibarra    | Sarah    | 987 Chen St.     | CA    | 3478679547 | SIbarra@gmail.com    | 2001-08-14 |        0 |   5
   21 | Frank     | Frank    | 231 Empire Ave.  | CA    | 3471112222 | Frank@gmail.com      | 2001-01-01 |       50 |   1
(21 rows)
```

Average of Numerical Field Procedure & Result

Now that I have shown the insert and delete procedures that were created within the database; I can also show the average procedure for the database. With this database I took the average of the amount of copies of all books found within the lirbary.

```
postgres=# CREATE OR REPLACE FUNCTION averageCopies()
postgres-# RETURNS float AS $average$
postgres$# declare
postgres$# average float;
postgres$# BEGIN
postgres$# RETURN(
postgres$# SELECT AVG(pnumcopy ) "Average of copies in library" FROM print
postgres$# );
postgres$# END;
postgres$# $average$
postgres-# LANGUAGE 'plpgsql';
CREATE FUNCTION
```

```
postgres=# SELECT averageCopies();
   averagecopies
--------------------
 3.739130434782610
(1 row)
```

With these procedures completed we can now move onto the trigger portion of the report. The trigger function receives data into their invoking environment through a structure named “TriggerData”. TriggerData contains a set of local variables that allow triggers to use numerous commands. The OLD and NEW commands represent the states of the row in the table before and after triggering the event to invoke the trigger. The next section will cover the trigger functions I implemented and created for the database.

Before Update Trigger

For this example, utilizing the BEFORE UPDATE command was important to accomplishing the task. This command will activate when any UPDATE to a record inside a table is invoked. This means that if a user were to attempt a change on the cardholder table then our update_trigger() would be called and executed to update any existing records of any columns that reference the column that the user is attempting to update.

```
postgres=# CREATE FUNCTION update_trigger() RETURNS trigger AS
postgres-# $BODY$
postgres$# BEGIN
postgres$# IF NEW.chID <> OLD.chID THEN
postgres$# UPDATE RentElectronic re
postgres$# SET OLD.chID = NEW.chID
postgres$# WHERE re.chID = c.chID;
postgres$# UPDATE RentPrint rp
postgres$# SET OLD.chID = NEW.chID
postgres$# WHERE rp.chID = c.chID;
postgres$# UPDATE Requests r
postgres$# SET OLD.chID = NEW.chID
postgres$# WHERE r.chID = c.chID;
postgres$#
postgres$# END IF;
postgres$#
postgres$# RETURN NEW;
postgres$# END;
postgres$# $BODY$
postgres-# LANGUAGE 'plpgsql';
CREATE FUNCTION
```

```
postgres=# CREATE TRIGGER beforeUpdate
postgres-# BEFORE INSERT OR UPDATE
postgres-# ON cardholder
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE update_trigger();
CREATE TRIGGER
postgres=#
```

Delete Trigger and Cascade Deletion

The BEFORE DELETE command is used to check if any row of data in cardholder has been deleted from the table. When a user attempts to delete a row from the cardholder table our trigger will also delete any references to that record in corresponding tables. This cascade delete allows the deletion of a row from cardholder without the complication of having to individually delete records in other tables.

```
postgres=# CREATE FUNCTION delete_trigger() RETURNS trigger AS
postgres-# $BODY$
postgres$# BEGIN
postgres$#
postgres$# DELETE FROM RentElectronic re
postgres$# WHERE re.chID = OLD.chID;
postgres$# DELETE FROM RentPrint rp
postgres$# WHERE rp.chID = OLD.chID;
postgres$# DELETE FROM Requests r
postgres$# WHERE r.chID = OLD.chID;
postgres$# END;
postgres$# $BODY$
postgres-# LANGUAGE 'plpgsql';
CREATE FUNCTION
```

```
postgres=# CREATE TRIGGER deleteRow
postgres-# BEFORE DELETE
postgres-# ON cardholder
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE delete_trigger();
CREATE TRIGGER
postgres=#
```

Instead-Of Trigger

The INSTEAD-OF triggers are special types that are row-level triggers that can return null to signal that an action was not performed during an update, and that the rest of the row can be skipped in the operation. INSTEAD-OF triggers can only be used in views.

First, I have to create a View to work with the INSTEAD-OF Trigger:

```
postgres=# CREATE OR REPLACE VIEW viewCardholders AS
postgres-# SELECT
postgres-# c.chID AS chID,
postgres-# c.lname AS lname,
postgres-# c.fname AS fname,
postgres-# c.address AS address,
postgres-# c.state AS state,
postgres-# c.phone AS phone,
postgres-# c.email AS email,
postgres-# c.bdate AS bdate,
postgres-# c.latefees AS latefees,
postgres-# c.lID AS lID
postgres-# FROM cardholder c;
CREATE VIEW
postgres=#
```

After the completion of the View; I'll show that it works by calling it. The viewCardholders View will display the cardholder table, which shows all the necessary information for a cardholder:

```
postgres=# select * from viewCardholders;
 chid |   lname   |  fname   |     address      | state |   phone    |        email         |   bdate    | latefees | lid
------+-----------+----------+------------------+-------+------------+----------------------+------------+----------+-----
    1 | Valle     | Jeff     | 415 Free St.     | CA    | 1421245123 | JValle@gmail.com     | 1967-10-04 |        0 |   1
    2 | Davidson  | Arthur   | 123 Nowhere Dr.  | CA    | 1217656654 | ADavidson@gmail.com  | 1978-02-01 |        5 |   2
    3 | Cameron   | James    | 7645 Camp St.    | CA    | 9871256546 | JCameron@gmail.com   | 1987-04-02 |        3 |   3
    4 | Hitchcock | Alfred   | 987 High Dr.     | CA    | 6551354766 | AHitchcock@gmail.com | 1990-06-07 |        0 |   4
    5 | Bradley   | David    | 988 Scott St.    | CA    | 7651243675 | DBradley@gmail.com   | 1998-07-09 |        0 |   5
    6 | Lynch     | David    | 3224 Empire Ave. | CA    | 4097092544 | DLynch@gmail.com     | 1998-06-06 |        1 |   1
    7 | Hardwick  | Chris    | 1234 Chen St.    | CA    | 1095346534 | CHardwick@gmail.com  | 1988-05-12 |        0 |   2
    8 | Wesker    | Albert   | 764 Sunrise Ave. | CA    | 4695431651 | AWesker@gmail.com    | 1987-07-06 |        0 |   3
    9 | Redfield  | Claire   | 731 Farm St.     | CA    | 5457641123 | CRedfield@gmail.com  | 1998-08-07 |      100 |   4
   10 | Acosta    | Derrick  | 654 Mega Ave.    | CA    | 3478766567 | DAcosta@gmail.com    | 1987-07-09 |        0 |   5
   11 | Burberry  | Jill     | 541 Scott St.    | CA    | 8976466567 | JBurberry@gmail.com  | 1989-02-12 |        5 |   1
   12 | Chucka    | Bill     | 542 Nowhere Dr.  | CA    | 4653766567 | BChucka@gmail.com    | 2001-05-03 |        0 |   2
   13 | Klein     | John     | 763 Camp St.     | CA    | 4521766567 | JKlein@gmail.com     | 1983-10-22 |        0 |   3
   14 | Stinson   | Daniel   | 431 High Dr.     | CA    | 3474356767 | DStinson@gmail.com   | 1992-11-15 |        5 |   4
   15 | Stein     | Alfred   | 772 Farm St.     | CA    | 3472315767 | AStein@gmail.com     | 1992-05-11 |        6 |   5
   16 | Scott     | Den      | 786 Empire Ave.  | CA    | 3476754367 | DScott@gmail.com     | 2002-03-15 |        2 |   1
   17 | Chin      | Jen      | 561 High St.     | CA    | 4567154316 | JChin@gmail.com      | 2001-10-15 |        0 |   2
   18 | Martinez  | Salvador | 873 Camp St.     | CA    | 3474378667 | SMartinez@gmail.com  | 1999-06-01 |        0 |   3
   19 | Hurtado   | Hector   | 980 Empire Ave.  | CA    | 3478438577 | HHurtado@gmail.com   | 1992-09-11 |        0 |   4
   20 | Ibarra    | Sarah    | 987 Chen St.     | CA    | 3478679547 | SIbarra@gmail.com    | 2001-08-14 |        0 |   5
   21 | Frank     | Frank    | 231 Empire Ave.  | CA    | 3471112222 | Frank@gmail.com      | 2001-01-01 |       50 |   1
(21 rows)
```

Now, I created an Instead-of trigger to work with our viewCardholders view shown here:

```
postgres=# CREATE FUNCTION update_view() RETURNS trigger AS
postgres-# $BODY$
postgres$# BEGIN
postgres$# UPDATE viewCardholders cd
postgres$# SET cd.chID = NEW.chID
postgres$# WHERE cd.chID = OLD.chID;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $BODY$
postgres-# LANGUAGE 'plpgsql';
CREATE FUNCTION
postgres=#
```

```
postgres=# CREATE TRIGGER instead_trigger
postgres-# INSTEAD OF UPDATE
postgres-# ON viewCardholders
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE update_view();
CREATE TRIGGER
postgres=#
```

Placing the created Triggers and Procedures/Functions into a Package

One main difference between postgreSQL and Oracle's PL/SQL is that Oracle makes use of an important function tool known as Package. Packages are a type of file that serve to contain a number of procedures or functions as well as triggers that a developer creates. Currently, postgreSQL does not have any functionality for Package creation. Due to this lack of support the discussion of packages will be continued as if they were implemented into postgreSQL.

Once our procedures, functions, and triggers are complete we would have the option to place them into a package. A package serves to group all user-created procedures, functions, and triggers into one single object. A package contains a “header” which contains the prototypes for all functions. It also includes the “body” which defines and implements all procedures and functions found within the package. In my own created package it would be useful for me to include all the previous procedures, functions, and triggers I made for the database. However, one important thing to look at while explain the next section is that PostgreSQL DOES NOT include packages, so there was no possible way for me to create such a thing if it is not supported.

If PostgreSQL supported package features; the database I would create for the database would look like this:

```
CREATE OR REPLACE PACKAGE cardholderInfo
IS
        PROCEDURE new_cardholder.sql;
        PROCEDURE delete_cardholder.sql;
        PROCEDURE averageCopies.sql;

        TRIGGER update_trigger.sql;
        TRIGGER beforeUpdate.sql;
        TRIGGER delete_trigger.sql;
        TRIGGER deleteRow.sql;
        FUNCTION update_view();
        TRIGGER instead_trigger.sql;
END;
```

4.8: PostgreSQL, Microsoft SQL Server, MySQL

The procedures and physical database were implemented using Postgres PL/pgSQL. However, there are many other commercial DBMS software that are popular and offer unique storage procedures and functionality. In the next section I will discuss the other widely used DBMS software such as Microsoft SQL-Server and MySQL. A look at how these other DBMS software handle procedures and syntax.

PL/pgSQL

The main differences between DBMS languages

We can begin this section by comparing postgreSQL and Oracle/MySQL. PostgreSQL and Oracle have advanced SQL capabilities that as a user of either the DBMS will need to learn specific commands for each database. PostgreSQL is considered to be much easier and simpler to use than MySQL, but both languages have great possibilities. Oracle has a much more diverse language and package implementation. Postgres allows developers to write code in multiple languages, so that they can implement any language they are familiar with into the database. This allows Postgres to have a much smoother implementation of language handling than MySQL

Postgres is an Open-Source software, so it lends itself to being a much more lightweight and easier software to use for developers. Postgres is easily attainable in any form, whether previous or current, by somebody who wishes to use it. Due to this, Postgres has large libraries readily available to developers. However, Oracle has some libraaries that are more polished than the ones found in Postgres. The Open-Source nature of Postgres versus Oracle's closed-source system is one of the major differences between the two DBMS languages.

Another difference between PostgreSQL and MySQL is the partitioning method used by MySQL which is called “MySQL cluster”. This allows MySQL to create multiple clusters within a single cluster instance within each individual node. PostgreSQL does not implement partitioning as complicated as MySQL.

Syntax for creating functions:

```
CREATE OR REPLACE FUNCTION func_example(parameter TYPE, parameter TYPE, …)
        RETURNS void AS $$
BEGIN
        INSERT INTO table_example VALUES (col1, col2, …);
END;
```

Syntax for a basic loop:

```
<<label>>
LOOP
        <statements;>
        EXIT [<<label>>] WHEN condition;
END LOOP;
```

Microsoft SQL Server: T-SQL

Comparisons to PostgreSQL DBMS Language

The main difference between both PL/pgSQL and Microsoft's SQL Server is that SQL Server does not include any functionality for CSV files. The COPY TO and COPY FROM commands are PL/pgSQL's implementation of CSV fiels. These commands are used to assist a developer with the use and importing/exporting of CSV files into their PostgreSQL database. Once large benefit to using MS T-SQL is that it declare variables in SQL scripts:

```
DECLARE @stuff INT = 5;
SELECT @stuff +6;
--returns 11
```

Compared to other DBMS languages, T-SQL offers developers the unique ability to open options for encrypting the text of procedures. T-SQL also offers encryption for restricting user defined permissions using the WITH Clause. T-SQL offers other benefits when compared to other DBMS languages, like multiple try-catch blocks in a procedure and that functions can easily return both scalar values nad tables.

Syntax for creating a Procedure:

```
CREATE { PROC|PROCEDURE } [schema_example.] procedure_example [ ; num ]
        [ { @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 creating a Function:

```
CREATE FUNCTION [ schema_example. ] function_example
( [ { @parameter_example [ AS ][ type_schema_name. ] parameter_data_type
        [ = default ] [ READONLY ] }
        [ ,...n ]
        ]
)
RETURNS return_data_type
        [ WITH <function_option> [ ,...n ] ]
        [ AS ]
         BEGIN
                function body
                RETURN scalar_example
        END
[ ; ]
```

Syntax for creating a basic loop:

```
WHILE @count < 50;
BEGIN
        {...statements...}
        SET @count = @count + 2;
END;
```

MySQL

Main comparisons between PL/pgSQL and MySQL

Both MySQL and PL/pgSQL are very similar in functionality, but it seems that MySQL is missing some features that postgreSQL has. MySQL does not offer a large amount of loops, because it only has WHILE-loops and basic loops implemented into itself. MySQL does not offer packages for namespace management, unlike postgreSQL. Also, the delimiter that ends statements in MySQL is "//" instead od the ";" in postgreSQL

Syntax for creating a procedure:

```
CREATE
        [ DEFINER = { user | CURRENT_USER } ]
PROCEDURE ab_example ( [ proc_example [ ,... ] ] )
        [ characteristic … ] routine_example
```

Syntax for a basic loop:

```
[ begin_label : ] WHILE search_example DO
        statement_example
END WHILE [ end_label ]
```

Oracle PL/SQL

Main comparisons between postgreSQL and PL/SQL

PL/SQL is the SQL-based language that Oracle uses to develop and implement physical databases. Oracle PL/SQL provides unique features when compared to other DBMS languages. The main feature is the package implementation which prevents name conflicts that add complex structures, like the "for" loop.

Syntax for creating a procedure:

```
CREATE OR REPLACE PROCEDURE <procedure example>
BEGIN
        <PL/SQL statements>
END
```

Syntax for creating a function:

```
CREATE OR REPLACE FUNCTION < function example>
RETURN
        <return type>
BEGIN
        <PL/SQL statements>
END
```

Syntax for basic loops:

```
WHILE <condition>
LOOP
        <statements>
END LOOP;

FOR <counter> IN  <range>
LOOP
        <statements<
END LOOP;
```

PHASE 5: Graphical User Interface Design and Implementation

Introduction

In the previous section, we discussed and implemented certain stored procedures, functions, triggers, and views to create functionality for the front end of a database. The next phase begins the implementation of a graphical user interface that connects with the created database. To begin, a list of the potential user groups must be examined and created in order for their needs to be met. I will also describe the implementation of procedures, functions, and triggers in the creation of the database. Screenshots will be provided for all the features in a portion of the application that is designed for one of the potential user groups. I will also provide information on the implementation process for the application, which will show portions of code for how the GUI works.

5.1: GUI application for Daily User Group and Functionality

To begin, a group of users must be specified in order to decide on how our GUI will function. So, the group that will be using the GUI needed for our database the most will be librarians. The database I have created was intended to be used by libraries to keep track of employees as well inventory, members of the library, and study rooms. All of which are managed by librarians within the library. So, primarily I will be focusing on the usage of the librarian, as the customer is going to be interacting with them the majority of the time. Another user group that will be focused on is the head librarian, who manages employees and inventory space. For the remainder of the Phase 5 section, I will be using my own implementation of the GUI for the librarian side of the application.

5.1.1: Librarian Side GUI

For the librarian side of the GUI application they will see the login page to determine which library they work at. They will be able to enter the site for their library and see a list of customers, books, electronics, and study rooms. They can then click on a member name to see their rental history. A librarian can also allow a customer to rent as book, or pay the late fees they may have accrued on their account.

Librarian Activities:

- Search through inventory of print/electronics
- See occupied/unoccupied study rooms
- Allow rentals of books/electronic/studyrooms to customers
- Allow customers to pay late fees
- Add customers to database

5.1.2: Head librarian Side GUI

For the purposes of the Head Librarian our database will start at the same log in page, but will have different privileges based on their level of status. A head librarian has all the same functionality of a regular librarian, but has the ability to add new employees and materials. Head Librarians will also be able to remove employees and members from the database.

Head Librarian activities:

- Add employees to database
- Add materials to the database
- Delete employees from the database
- Delete members from the database

5.1.3: Itemized Description of GUI Application

For the library database that I created the items that would be largely dealt with were books that were donated to the library as well as rented out by customers. Books have a unique key without having to create one in a database with the use of the ISBN. The books and how many different books we have in our database is the main focus of the database, as books are used in nearly every action found within our database. Librarians can search for books, see a list of books, see a list of books that have been rented by a single person, and rent out books to customers in the front-end application of our database. Books also have general information associated with them like title, publish date, and author.

5.1.4: Screenshots, Description Functionalities, and content of Data

As explained before, the main focus of our database items will be books that are rented out to customers. I will now demonstrate and explain different portions of the created GUI application through screenshots of the website that was created for the database. A brief description will follow each screenshot that is provided

List of Books:

ID	ISBN	Publisher	Issue Date	Copies	Author	Library	Title	Year	Genre
1	2345121153	White Wolf	1986	4	Steven King	1	IT	1986	Horror
2	1231234513	Giant	1949	3	George Orwell	1	1984	1949	Science Fiction
3	5664272312	Adventure Inc.	1985	4	Steven King	1	Carrie	1985	Horror
6	54837458123	Hello Books	1978	4	Steven King	1	The Stand	1978	Science Fiction
11	983248234	Yellow Inc	1952	3	Ralph Ellison	1	The Invisible Man	1952	Horror
17	8976737818	Giant	1998	5	Duponte James	1	There Be Dragons	1998	Fantasy

This photo currently shows the list of books found in one of three libraries which shows the ISBN, publisher, publisher date, copies in the library, author, which library it is in, title, and genre.

List of Members:

ID	Last Name	First Name	Address	State	Phone	E-mail	Date of Birth	Fees	Library
1	Valle	Jeff	415 Free St.	CA	1421245123	JValle@gmail.com	1967-10-04	0	1
16	Scott	Den	786 Empire Ave.	CA	3476754367	DScott@gmail.com	2002-03-15	2	1
6	Lynch	David	3224 Empire Ave.	CA	4097092544	DLynch@gmail.com	1998-06-06	0	1
11	Burberry	Jill	541 Scott St.	CA	8976466567	JBurberry@gmail.com	1989-02-12	0	1

This screenshot shows the list of members at one of three libraries which shows each members first name, last name, ID in the library, contact information, and any fees they have with that library.

List of Electronics

ID	Production	Visual	Copies	Library	Title	Year	Genre
5001	White Wolf	t	3	1	One Flew Over the Cuckoos Nest	1975	Fiction
5006	Fox	f	3	1	Full Metal Jacket	1987	Fiction
5011	White Wolf	f	2	1	Boxed	1988	Horror
5017	Edison	t	3	1	Space Odyssey	2004	Science Fiction

This screenshot shows the list of electronics available at one of three libraries which, like the book list, has the production company, title, ID in library, number of copies, and genres.

List of StudyRooms:

ID	Capacity	Purpose	Location	Library
1	4	Study	203	1
6	2	Study	210	1
11	5	Study	201	1
16	4	Study	105	1
21	4	Visual Room	204	1

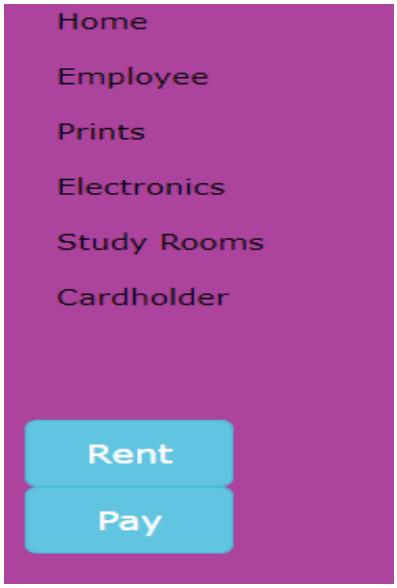
This screenshot shows the list of study rooms available at one of three libraries in the database, which shows the capacity, purple, and floor location in the library that the room is located.

List of Employees:

ID	Last Name	First Name	SSN	Address	State	Phone	E-mail	Date of Birth	Payrate	Position	Library
1	Harriet	William	575236741	312 What St.	CA	6612311233	WHarriet@gmail.com	1982-12-03	12	Library Technician	1
2	Jones	Tanya	656326572	321 No St,	CA	6611322315	TJones@gmail.com	1998-04-13	0	Volunteer	1
8	Tuck	James	564126713	512 Juniper Dr.	CA	8552136587	JTUCK@gmail.com	1985-08-03	12	Library Technician	1
9	Sam	Neil	124231557	123 Capree St.	CA	1511244512	NSamgmail.com	1987-05-02	40	Head Librarian	1
14	Gates	Sally	677897641	924 Juniper Dr.	CA	4512325411	SGates@gmail.com	1993-01-16	40	Head Librarian	1

This screenshot shows the list of employees at a single library that shows information for employees like contact information, job position, and pay rate.

Sidebar Buttons:

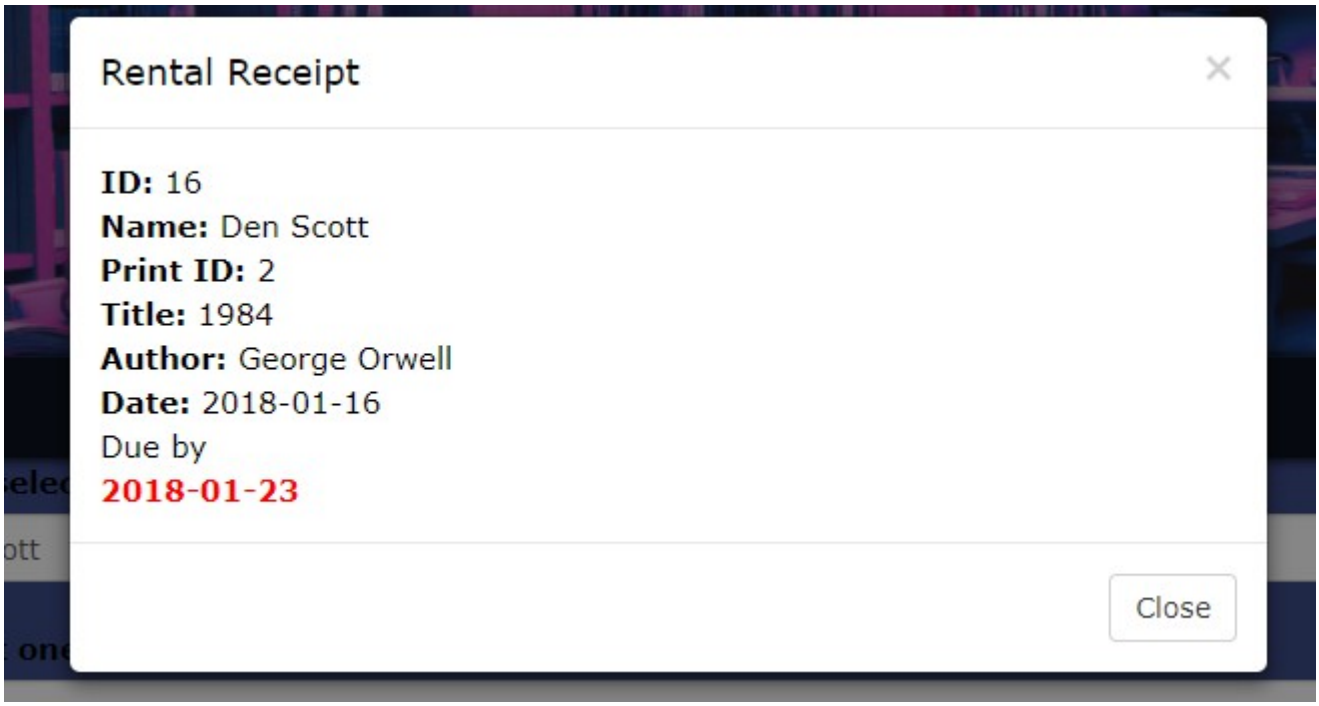


This screenshot shows the buttons that are usable for the GUI for one library, each button is functional. Rent will allow a librarian to choose a member and book for that member to rent. It will also allow a librarian to let a customer pay their overdue fees for books.

Rent Button:

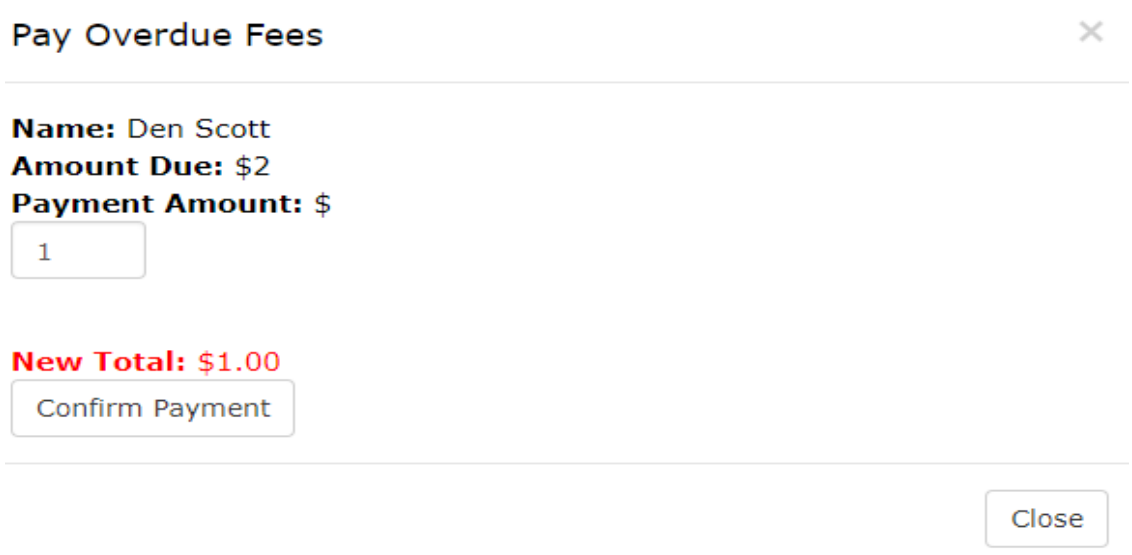
This screenshot displays the drop down menu that a librarian can use to select a member and book to rent out to a customer. Once the correct parameters are found pressing the button will bring up the receipt.

Rent Receipt:



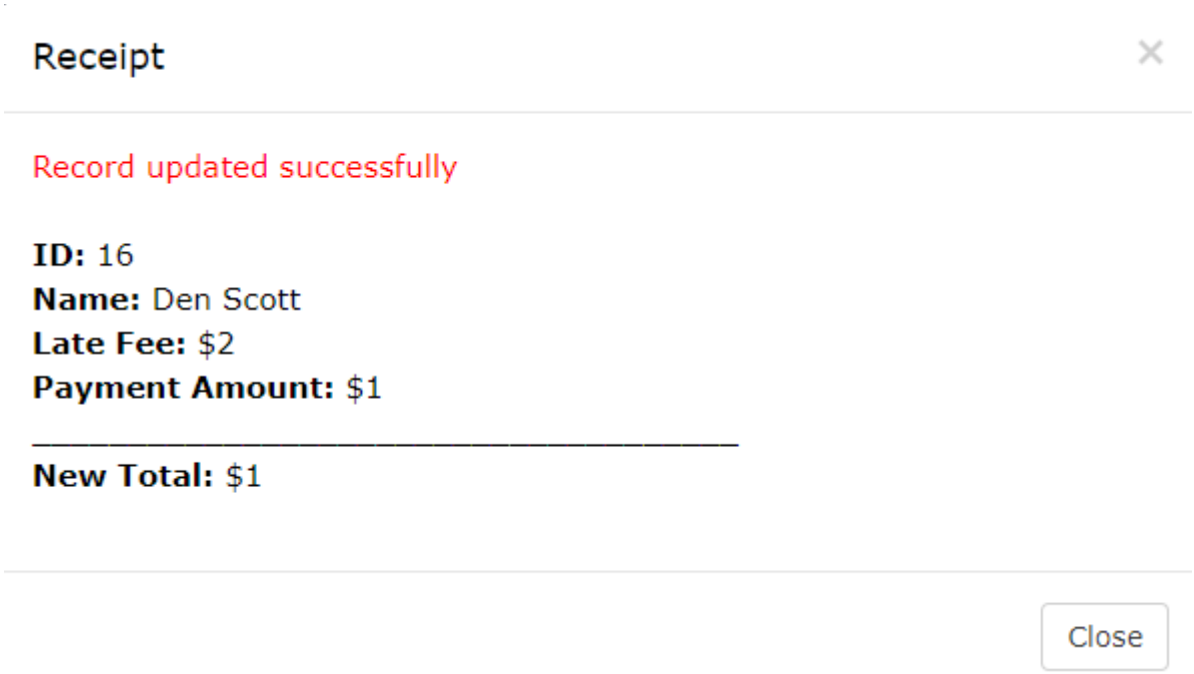
This is the receipt given to a member after they have rented a book which shows the ID of the person it was rented to, the ID of the book, and the date it was rented, as well as the due date for returns.

Overdue fees:



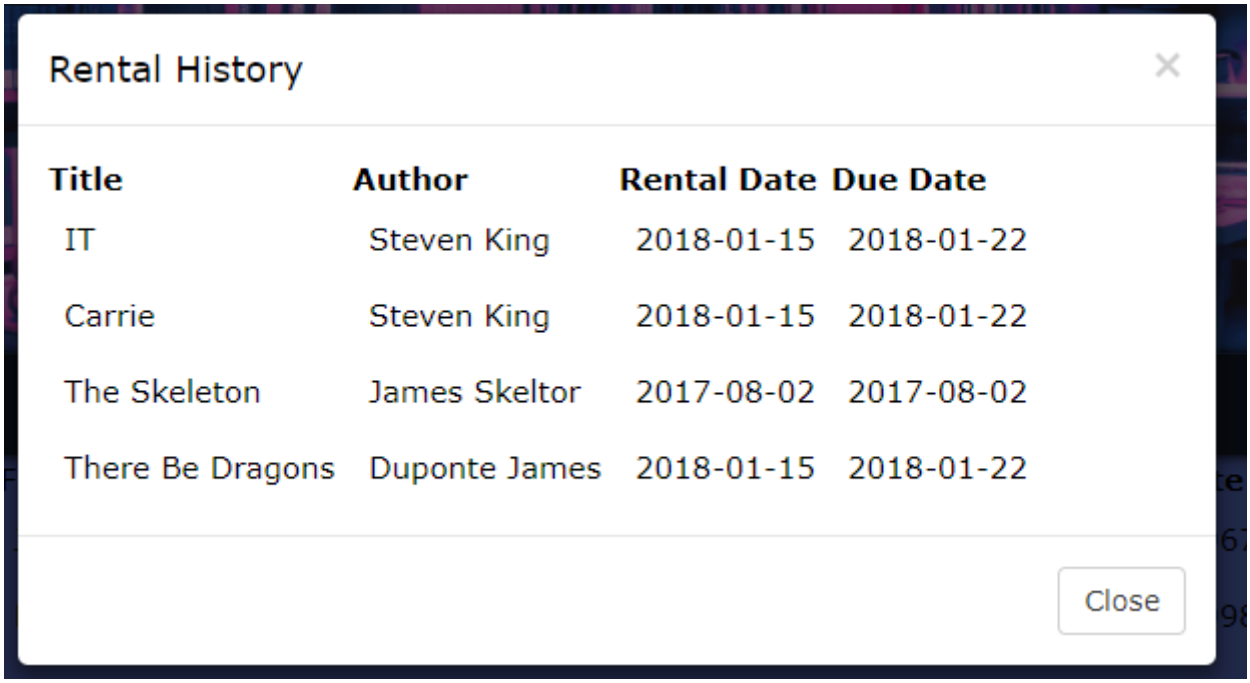
This screenshot shows the amount of money a person can pay towards their rental fees, and then a button to allow a customer to finish the transaction and receive a receipt.

Receipt for payment:



After a transaction has occurred a receipt will pop up that shows the amount paid by the customer, how much they owed before the payment, and how much they owe after the payment.

History of Rent:



If a members account is clicked in the cardholder menu a display of their entire rental history is shown so we can see how many books they have rented throughout their membership.

Photo of Content Data:

```
postgres=# SELECT * FROM print;
 printid |    isbn     |   publisher    | issuedate | pnumcopy |       author       | lid |       title       | year |      genre
---------+-------------+----------------+-----------+----------+--------------------+-----+-------------------+------+-----------------
       1 |  2345121153 | White Wolf     | 1986      |        4 | Steven King        |   1 | IT                | 1986 | Horror
       2 |  1231234513 | Giant          | 1949      |        3 | George Orwell      |   1 | 1984              | 1949 | Science Fiction
       3 |  5664272312 | Adventure Inc. | 1985      |        4 | Steven King        |   1 | Carrie            | 1985 | Horror
       4 |  5664272312 | Adventure Inc. | 1985      |        3 | Steven King        |   2 | Carrie            | 1985 | Horror
       5 |  5555121328 | White Wolf     | 1932      |        3 | Aldous Huxley      |   3 | Brave New World   | 1932 | Science Fiction
       6 | 54837458123 | Hello Books    | 1978      |        4 | Steven King        |   1 | The Stand         | 1978 | Science Fiction
       7 | 54837458123 | Hello Books    | 1978      |        5 | Steven King        |   5 | The Stand         | 1978 | Science Fiction
       9 |  4124125563 | Giant          | 1990      |        3 | Mechael Crichton   |   5 | Jurassic Park     | 1990 | Science Fiction
      10 |   678562132 | Goodbye Books  | 2011      |        4 | Ernest Cline       |   2 | Ready Player One  | 2011 | Science Fiction
      11 |   983248234 | Yellow Inc     | 1952      |        3 | Ralph Ellison      |   1 | The Invisible Man | 1952 | Horror
      12 |   983248234 | Yellow Inc     | 1952      |        2 | Ralph Ellison      |   3 | The Invisible Man | 1952 | Horror
      13 |   123182848 | Edison         | 1954      |        5 | J.R.R. Tolkien     |   4 | Lord of the Rings | 1954 | Fantasy
      14 |  1294737818 | Hello Books    | 1925      |        4 | F. Scott Fitzgerald|   4 | The Great Gatsby  | 1925 | Fiction
      15 |  1294737818 | Hello Books    | 1925      |        3 | F. Scott Fitzgerald|   4 | The Great Gatsby  | 1925 | Fiction
      16 |  1342337818 | White Wolf     | 1940      |        3 | James Skeltor      |   4 | The Skeleton      | 1940 | Horror
      17 |  8976737818 | Giant          | 1998      |        5 | Duponte James      |   1 | There Be Dragons  | 1998 | Fantasy
      18 |  1299807818 | Edison         | 1990      |        4 | Scott Steiner      |   2 | All The Math      | 1990 | Biography
      19 |  1294737865 | Goodbye Books  | 2001      |        4 | Jack Skunk         |   5 | A Stinky Story    | 1998 | Non-Fiction
      20 |  1675737818 | Yellow Inc     | 1967      |        5 | Thomas James       |   4 | Box Boy           | 1967 | Fantasy
      21 |  1294897518 | White Wolf     | 1987      |        4 | John Sacks         |   4 | The Potato Bag    | 1990 | Non-Fiction
      22 |  7654237818 | Giant          | 2008      |        3 | Pete Passer        |   3 | Big Pumpkin       | 2008 | Horror
      23 |  1294787650 | Adventure Inc  | 2005      |        4 | Harold Ramis       |   5 | Buster            | 2005 | Fiction
      24 |  1289637818 | Hello Books    | 2004      |        4 | Dave Buster        |   3 | The Big Bomb      | 2004 | Science Fiction
(23 rows)
```

```
postgres=# SELECT * FROM cardholder;
 chid |   lname   |  fname   |      address      | state |   phone    |        email         |   bdate    | latefees | lid
------+-----------+----------+-------------------+-------+------------+----------------------+------------+----------+-----
    1 | Valle     | Jeff     | 415 Free St.      | CA    | 1421245123 | JValle@gmail.com     | 1967-10-04 |        0 |   1
    2 | Davidson  | Arthur   | 123 Nowhere Dr.   | CA    | 1217656654 | ADavidson@gmail.com  | 1978-02-01 |        5 |   2
    3 | Cameron   | James    | 7645 Camp St.     | CA    | 9871256546 | JCameron@gmail.com   | 1987-04-02 |        3 |   3
    4 | Hitchcock | Alfred   | 987 High Dr.      | CA    | 6551354766 | AHitchcock@gmail.com | 1990-06-07 |        0 |   4
    5 | Bradley   | David    | 988 Scott St.     | CA    | 7651243675 | DBradley@gmail.com   | 1998-07-09 |        0 |   5
    6 | Lynch     | David    | 3224 Empire Ave.  | CA    | 4097092544 | DLynch@gmail.com     | 1998-06-06 |        1 |   1
    7 | Hardwick  | Chris    | 1234 Chen St.     | CA    | 1095346534 | CHardwick@gmail.com  | 1988-05-12 |        0 |   2
    8 | Wesker    | Albert   | 764 Sunrise Ave.  | CA    | 4695431651 | AWesker@gmail.com    | 1987-07-06 |        0 |   3
    9 | Redfield  | Claire   | 731 Farm St.      | CA    | 5457641123 | CRedfield@gmail.com  | 1998-08-07 |      100 |   4
   10 | Acosta    | Derrick  | 654 Mega Ave.     | CA    | 3478766567 | DAcosta@gmail.com    | 1987-07-09 |        0 |   5
   11 | Burberry  | Jill     | 541 Scott St.     | CA    | 8976466567 | JBurberry@gmail.com  | 1989-02-12 |        5 |   1
   12 | Chucka    | Bill     | 542 Nowhere Dr.   | CA    | 4653766567 | BChucka@gmail.com    | 2001-05-03 |        0 |   2
   13 | Klein     | John     | 763 Camp St.      | CA    | 4521766567 | JKlein@gmail.com     | 1983-10-22 |        0 |   3
   14 | Stinson   | Daniel   | 431 High Dr.      | CA    | 3474356767 | DStinson@gmail.com   | 1992-11-15 |        5 |   4
   15 | Stein     | Alfred   | 772 Farm St.      | CA    | 3472315767 | AStein@gmail.com     | 1992-05-11 |        6 |   5
   16 | Scott     | Den      | 786 Empire Ave.   | CA    | 3476754367 | DScott@gmail.com     | 2002-03-15 |        2 |   1
   17 | Chin      | Jen      | 561 High St.      | CA    | 4567154316 | JChin@gmail.com      | 2001-10-15 |        0 |   2
   18 | Martinez  | Salvador | 873 Camp St.      | CA    | 3474378667 | SMartinez@gmail.com  | 1999-06-01 |        0 |   3
   19 | Hurtado   | Hector   | 980 Empire Ave.   | CA    | 3478438577 | HHurtado@gmail.com   | 1992-09-11 |        0 |   4
   20 | Ibarra    | Sarah    | 987 Chen St.      | CA    | 3478679547 | SIbarra@gmail.com    | 2001-08-14 |        0 |   5
   21 | Frank     | Frank    | 231 Empire Ave.   | CA    | 3471112222 | Frank@gmail.com      | 2001-01-01 |       50 |   1
(21 rows)
```

In these photos you can see the full list of books and employees within our database. This is just two tables found within our database, which contains even more information.

5.1.5: Tables, Views, Stored Subprograms, and Triggers

To have a successful graphical user interface that works properly with a database I had to use the built in functionalities found within PostgreSQL. Some of these functions are known as Views, as well as Stored procedures and functions that were previously mentioned in Phase 4. The next sections will discuss the back end of the GUI and how postgreSQL was utilized in the creation of the front end.

Tables:

```
             List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | cardholder      | table | postgres
 public | electronic      | table | postgres
 public | employee        | table | postgres
 public | library         | table | postgres
 public | print           | table | postgres
 public | rentelectronic  | table | postgres
 public | rentprint       | table | postgres
 public | requests        | table | postgres
 public | studyrooms      | table | postgres
 public | worklog         | table | postgres
 public | worksfor        | table | postgres
(11 rows)
```

The physical database that was created consists of 12 tables that all have contents of data stored within them. Creating both relations and relationship relations allows the database to be fully connected and meaningful. As shown in the previous tables, print, and employee.

View:

For viewing a list of the appropriate content for specific users I used database queries that are found within the snippets of code, and specific to the user's needs.

Delete:

The deletions I would have used if necessary would have been performed using the DELETE procedure and querying into the database.

Insert:

Insert was used numerous times along with a query in order to add books to rental histories, and adding to forms when they are changed by the librarian user.

Update:

The update procedure is used to update the existing information of a member when they choose to pay any late fees they have accrued through the library.
Trigger: update_trigger()
This trigger is used when an update is made to cardholders, such as a name change, address change, or fee change. Anything that is referencing cardholders also gets updated.

Trigger: delete_trigger()
This trigger works similar to update, in that it updates all tables referencing cardholder. However, it only does so when an attempt to delete is used against the table.

5.2: Programming Sections

In the next section I will give a brief overview of how the user is able to add, update, and make orders through the front end application. Code will be shown here with a brief explanation as to what it does.

5.2.1: Server-Side Programming

This section will give a brief explanation of how the functions in the GUI work, such as the numerous buttons on the page, as well as the checking of rental histories.

PHP and HTML

To begin, much of the graphical user interface was created using HTML and Javascript, and used PHP to connect these programs to the database that was created for the project. PHP largely dealt with communication from database to GUI, while HTML and Javascript focused on GUI related issues.

```
if (str == "print") {
  tablestr += "<tr><th>ID</th><th>ISBN</th><th>Publisher</th><th>Issue Date</th><th>Copies</th><th>Author</th><th>Library</th><th>Title</th><th>Year</th><th>Genre</th></tr>";
}
if (str == "electronic") {
  tablestr += "<tr><th>ID</th><th>Production</th><th>Visual</th><th>Copies</th><th>Library</th><th>Title</th><th>Year</th><th>Genre</th></tr>";
}
if (str == "studyrooms") {
  tablestr += "<tr><th>ID</th><th>Capacity</th><th>Purpose</th><th>Location</th><th>Library</th></tr>";
}
if (str == "cardholder") {
  tablestr += "<tr><th>ID</th><th>Last Name</th><th>First Name</th><th>Address</th><th>State</th><th>Phone</th><th>E-mail</th><th>Date of Birth</th><th>Fees</th><th>Library</th></tr>";
}
tablestr += this.responseText;
tablestr += "</table>";
// document.getElementById("content").innerHTML += this.responseText;
document.getElementById("content").innerHTML = tablestr;
```

This code shows a portion of the homepage's sidebar that a user can use to navigate from page to page. You can see the tags print, electronic, studyrooms, and cardholder.

Button Functionality:

```
<li><button type="button" class="btn btn-info btn-lg" onclick="rentPrint()" style="width:100px;">Rent</button></li>
<li><button type="button" class="btn btn-info btn-lg" onclick="payOverdue()" style="width:100px;">Pay</button></li>
</ul>
```

```
xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      rent = "<div class=\"form-group\"><label for=\"sel1\">Renter ID (select one):</label><select class=\"form-control\" id=\"renter\">";
      rent += this.responseText;
      rent += "</div><button type=\"button\" class=\"btn btn-info btn-lg\" onclick=\"rentTransaction()\" data-toggle=\"modal\" data-target=\"#rentModal\">Rent Print</button>";
      document.getElementById("content").innerHTML = rent;
    }
```

Both of these code snippets show buttons for different reasons, one is a button the homepage that is used to rent a book (rentPrint) and another is rentPrint that allows a user to rent a book through the button rentTransaction. There are many buttons in the GUI that are fully functional.

Order History:

```
// Order History
elseif(is_null($t)){
  $query = "SELECT print.title, print.author, rentprint.prstart, rentprint.prend
            FROM print, cardholder, rentprint
            WHERE cardholder.chID =".$cid. " AND rentprint.chID = cardholder.chID AND rentprint.printID = print.printID
            AND cardholder.lID=".$id." ORDER BY rentprint.prstart;";
  $result = pg_query($query) or die('Query failed' . pg_last_error());
  echo "<table><tr><th>Title</th><th>Author</th><th>Rental Date</th><th>Due Date</th></tr>\n";
  while($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col) {
      echo "\t\t<td>";
      echo $col;
      echo "</td>\n";
    }
    echo "\t</tr>\n";
  }
  echo"</table>\n";
  pg_free_result($result);
}
```

This screenshot shows how a query is made to the database when a users name is clicked to see their entire rental history. This is used to display the query to the GUI for the librarian.

```
// Query
$t = $_GET['t'];
$id = $_GET['id'];
$cid = $_GET['cid'];
$rent = $_GET['rent'];
$pay = $_GET['pay'];
$addpay = $_GET['addpay'];
$newtotal = $_GET['newtotal'];
$oldtotal = $_GET['oldtotal'];
$paytotal = $_GET['paymenttotal'];
$prstart = $_GET['prstart'];
$prend = $_GET['prend'];
$pid = $_GET['pid'];
$renttransaction = $_GET['renttransaction'];
```

This screenshot is the code that finds the specific user that is necessary to make the above picture work. A member ID is passed to this Query and the above picture links this data to display.

Insert Book for Rental:

```
pg_free_result($result);
echo "</select><br><label for=\"sel1\">Print (select one):</label><select class=\"form-control\" id=\"print\">";
$query = "SELECT printID, title, author FROM print WHERE lid='".$id."' AND pnumcopy > 0 ORDER BY title;";
$result = pg_query($query) or die('Query failed' . pg_last_error());
while($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
  $print = array();
  foreach ($line as $col) {
    $print[] = $col;
  }
  echo "<option value=\"" . $print[0] . "\">";
  echo $print[1] . " - " . $print[2];
  echo "</option>";
}
echo "</select>";
pg_free_result($result);
}
```

This Screenshot shows the list of books in the database in a drop down menu for librarians that wish to assign a book to a customers.

Rental Receipt:

```
$query = "SELECT title, author FROM print WHERE printID=".$pid." ORDER BY printID;";
pg_free_result($result);
$result = pg_query($query) or die('Query failed' . pg_last_error());
while($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
  $print = array();
  foreach ($line as $col) {
    $print[] = $col;
  }
  echo "<br><b>Print ID:</b> ". $pid;
  echo "<br><b>Title:</b> ".$print[0];
  echo "<br><b>Author:</b> ".$print[1];
}
echo "<br><b>Date: </b>". $prstart;
echo "<br>Due by <b><div style=\"color:red\">" .$prend ."</div></b>";
pg_free_result($result);
```

After a rental has been processed , this code will show the receipt for the transaction that includes the due by date and well as the ID of the person who rented it out.

Payment:

```
// Late Fee Payments
elseif($pay == 1){
  $query = "SELECT chID, fname, lname, latefees FROM cardholder WHERE latefees > 0 AND lid='".$id."' ORDER BY chID;";
  $result = pg_query($query) or die('Query failed' . pg_last_error());
  echo "<table><tr><th>ID</th><th>First</th><th>Last</th><th>Overdue Fees</th></tr>\n";
  while($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
    echo "\t<tr data-toggle=\"modal\" data-target=\"#payment\" onclick=\"payModal(";
    $x = 0;
    foreach ($line as $col) {
      if($x == 0){
        echo $col;
        echo ")\">\n\t\t<td>";
        echo $col;
        echo "</td>\n";
        $x++;
      }
      else{
        echo "\t\t<td>";
        echo $col;
        echo "</td>\n";
      }
    }
    echo "\t</tr>\n";
  }
  echo"</table>\n";
  pg_free_result($result);
}
```

```php
// Change fee total after payment
elseif($addpay == 1){
  $update = "UPDATE cardholder SET latefees=" .$newtotal. " WHERE chID=".$cid;
  // if ($db_connection->query($update) === TRUE) {
  //   echo "Record updated successfully";
  // } else {
  //   echo "Error updating record: " . $db_connection->error;
  // }
  $result = pg_query($db_connection, $update);
  if (!$result) {
    echo "Error updating record: " . $db_connection->error;
    exit;
  } else {
    echo "<div style=\"color:red\">Record updated successfully</div><br>";
    $query = "SELECT chID, fname, lname, latefees FROM cardholder WHERE chID=". $cid;
    $result = pg_query($query) or die('Query failed' . pg_last_error());
    $cardholder = array();
    while($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
      foreach ($line as $col) {
        $cardholder[] = $col;
      }
    }
    echo "<b>ID:</b> ". $cardholder[0];
```

These two pictures show the list of users who must make a payment to the library and the changes to the list after a payment has been made, if a person fully pays their debt to the library they are removed from the list.

PaymentReceipt:

```php
echo "<div style=\"color:red\">Record updated successfully</div><br>";
$query = "SELECT chID, fname, lname, latefees FROM cardholder WHERE chID=". $cid;
$result = pg_query($query) or die('Query failed' . pg_last_error());
$cardholder = array();
while($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
  foreach ($line as $col) {
    $cardholder[] = $col;
  }
}
echo "<b>ID:</b> ". $cardholder[0];
echo "<br><b>Name:</b> " . $cardholder[1] . " " . $cardholder[2];
echo "<br><b>Late Fee:</b> $" . $oldtotal;
echo "<br><b>Payment Amount:</b> $" . $paytotal;
echo "<br>_____";
echo "<br><b>New Total:</b> $" . $cardholder[3];

g_free_result($result);
```

This code shows the full receipt of that account that has paid some amount of money to the library, and will update with a new total after calculations are done in the database.

5.2.2: Middle-Tier Programming

This section will be showcasing the code that communicates between PHP and HTML that creates the connection to the physical database and allows the display of information that is used for the GUI.

Basic Connection:

To connect to a physical database to HTML, a developer would have to use a PHP statement that allows such a connection to even happen, as shown here.

```php
// Connecting, selecting database
$db_connection = pg_connect("host=localhost dbname=jmartinez3 user=jmartinez3 password=ztirq4Nild")
  or die('Could not connect to database' . pg_last_error());
```

In my database, I created a variable called $db_connection in order to establish a connection between PHP and the physical database. This is established by using a username and password that verifies it is the correct database you wish to pull information from.

Basic Queries:

```php
$query = "SELECT * FROM ".$t." WHERE lid='".$id."'";
$result = pg_query($query) or die('Query failed' . pg_last_error());
while($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
```

Once a connection is established, the user can make simple queries to the database using PHP. This is done by making a variable like $book, and then initializing its value to the query that you need. This is what allows a developer to create a connection to a physical database.

5.2.3: Client-Side Programming

This section will go over the importance of the other languages used to create functionalities for the graphical user interface that worked alongside HTML and PHP to create a complete database. Much of the code on the front end was created to help create an environment that was enjoyable to look at as well as easy to use, and this is due to the tools that are available to developers. HTML is one of these tools, and allowed the front end to feel much more organized and complete. Much of the HTML and PHP was learned from w3-schools and other tutorials. Another program that was used for this front end was the inclusion of Javascript and Ajax. These two tools used together allowed the use of their individual functionalities to come together and form a complete website.

```
function payModal(chid){
  recURL = 'receipt.php?id=1&chid=' + chid;
  $.ajax({
    url: recURL,
    dataType: 'json'
  }).done(
    function(data){
      var fname = data[0];
      var lname = data[1];
      var fees = data[2];
      payform = "<b>Name:</b> " + fname + " " + lname;
      payform += "<br><b>Amount Due:</b> $" + fees;
      payform += "<br>";
      payform += "<b><div id=\"payname\">Payment Amount:</b> $<div class=\"form-group row\">
                  <div class=\"col-xs-2\"><input class=\"form-control\" id=\"paid\"
                 type=\"text\" onchange=\"paidMath(" + fees + "," + chid + ")\"></div></div>";
      document.getElementById("pay").innerHTML = payform;
      document.getElementById("newtotal").innerHTML = "";
      document.getElementById("receipt").innerHTML = "Pay Overdue Fees";
    });
}
```

In this example I used this Ajax call to request data from the database back to the Javascript function payModal( ) to populate the modal with the receipt information. The Ajax call is sent to the receipt.php file and sends that information to be attached and displayed on the Javascript function. This is done in the php file by creating an array of the data to be sent back to the Javascript function and convert the array into JSON using the php function json_encode( ).

5.3: Concluding Review

Before I end this report, a section that allows comments and lessons learned throughout the developer process is essential in the process of learning. In this section a final review will take place for all the steps of designing, implementing, and programming a database.

Information Collection and Analysis:

In this section it is important to gather all information needed for a developers database, any problems that exist in this portion of database creation will persist in other phases of the complete project. Gathering information from a business is both important and necessary, all options must be exhausted to fully meet the needs of all types of users.

Conceptual Database Design:

In this portion of the design process we are required to use the previous data collection to create a conceptual database. This is formed in the Entity-Relationship Diagram, and this will display the entities and their relationships to each other. This section is truly the most important part of designing a database, any mistakes in the previous section should be solved at this point in the process. If I were able to, I would change many issues I have with my own database, and many of those issues stem from not understanding how crucial this phase was at the time of designing the database.

Logical Database Design:

During this phase the process of creating a logical database design from a conceptual database design begins. In this section we intend to represent the database mathematically through the use of Relational Algebra, Tuple Relational Calculus, and Relational Calculus. This process requires accuracy and precision in order to allow the next phase to go smoothly.

Physical Database Implementation:

In this phase a DBMS is chosen according to its functionality in order to create a physical database. The database needs to contain real-world information in order to be considered a success. The developer has the option to create procedures, triggers, and functions in this step to get better functionality from the physical database.

Database Application:

The final phase of the database design process a front end application is designed for specific user groups in order for the physical database to be easily accessed and manipulated. This portion of the design process must be easy to use and simple enough for any user to fully comprehend what they are doing. Administrators and regular users should be able to easily manage and manipulate data through the use of a well designed graphical user interface.

5.3.1: Embedded Survey Questions

Outcome	Jeremy Martinez
An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution	8
An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs. An ability to understand the analysis, design and implementation of a computerized solution to a real-like problem.	9
An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software specification white paper or user manual.	9
An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer system in a way that demonstrates comprehension of the trade offs involved in design decisions.	7