Landscape Service Database

Erick Toscano

Eddie Velasco

CMPS 3420 - Fall 2018

California State University, Bakersfield

Table of Contents

Phase 1: Fact-Finding, Information Gathering, and Conceptual Database Design

1.1 Fact-Finding Techniques and Information Gathering

In order to build the database for an organization, a database designer must have a thorough comprehension of the organization they are hired to design. The following section will give the introduction to the organization, an explanation of the research process, the scope of the database design, and itemized descriptions of the entity and relationship sets.

1.1.1 Introduction to Enterprise/Organization

Green Landscaping is a fictitious landscaping service designed to represent aspects of how Toscano Landscaping, a local landscaping business, handles their work flow. Green Landscaping offers landscaping services to residential homes on a weekly basis. Services are offered in the form of contracts with defined terms or one-time service work. The owner or manager of the business is in charge of keeping track of all contracts, expenses, income, and employee payroll.

1.1.2 Description of Fact-Finding Techniques

Fact-finding for the Green Landscaping database relied much on the general understanding of landscaping businesses as well as information provided by Toscano Landscaping which is owned and operated by an immediate family member. We verified facts and gathered information through general talks and inquiries of how Toscano Landscaping ran their business. This allowed us to design a database that fully captured the intended business model. By interviewing the appropriate and potential users of the database we formed a well-thought out structure for our database. From contracts to tools and so on, we could observe and conceptualize the regular logistics of how the business was managed.

1.1.3 Database Design Scope

An important aspect when developing the concept of a database is to know what segments of the organization will be represented by the database. We call the real world aspects the miniworld or universe of discourse. This is important so we can keep in mind the potential users and applications our database will supply. Landscaping businesses keep written documents to track all the contracts, services,

client information, and employee information. They also require employees to have access to routes, addresses, work vehicles, and tools. The goal for the database we will design for our business, Green Landscaping, will be to create an efficient and easy to use system for both owner and employee interactions. An effective model will enable us to implement features and applications that will help us in our goal.

1.1.4 Entity and Relationship Set Description

After all our information and facts are gathered, the data can be represented as entity sets and relationship sets as is described in a typical E-R model. In the remainder of this section, these entities and relationships will be described. A further detailed explanation will be provided in section 1.2.

Entity Type Definitions

Income: Income_ID, Date, Type, Amount, Description
An **income** is a record of a payment received to the business.

Contract: Contract_ID, Price, Description
A **contract** is a document written between the business and a client for services.

House: House_ID, Address, Start_Date, End_Date
A **house** is the centralized location for services that will be rendered to client.

Additional Service: Service_ID, Price, Description
An **additional service** is a service not specified under contract but performed for a house.

Client: Client_ID, Name, Address, Phone Number
A **client** is an individual who owns the house where services will be rendered.

Route: Route_ID, Start_Date, End_Date
A **route** is a designated list of houses in which employees will follow to perform services.

Car: Car_ID, Start_Date, End_Date, Car_Desc
A **car** is a vehicle used by an employee or employees to transport tools and perform services.

Tool: Tool_ID, Brand, Name, Description
A **tool** is a piece of equipment used by an employee kept in a car that will be used to landscape.

Employee: Employee_ID, Name, Address, Phone, Salary
An **employee** is an individual who works for the business, performs work, and is paid.

Payment: Payment_ID, Amount, Date, Type, Hrs_Worked, Description
A **payment** is a record of a payment given to an employee.

Relationship Type Definitions

Employee **paid** by Payment; Cardinality: 1..N; Participation: Total, Total

Employee **assigned to** Car; Cardinality: N..1; Participation: Total, Total

Car **contains** Tools; Cardinality: 1..N; Participation: Total, Total

Route **worked by** Car; Cardinality: 1..1; Participation: Total, Total

House **assigned** to Route; Cardinality: 1..1; Participation: Total, Total

House **owned by** Client; Cardinality: N..1; Participation: Partial, Total

Additional Service **provided to** House; Cardinality: N..1; Participation: Partial, Total

Income **produced by** Contract; Cardinality: N..1; Participation: Total, Total

1.2 Conceptual Database Design

In this section we will discuss the conceptual design to Green landscaping database. We will describe its entities and corresponding attributes and some operations that will be supported by this model.

Before we design a fully operational database we must first implement a conceptual design so we can understand how the data will be stored. In this example we will be using the Entity Relationship (ER) model to understand the behavior of our database.

In this model we will describe an entity as an object, such as employee or client; with these entities will be attributes which will describe qualities of the entity. We also create relationships between entities which will describe how these objects are associated in relation to the database.

We must understand this blueprint of the database, or schema, which should not change too often since it is a description of the database occurrences before they have been initiated, so we can firmly present a diagram that visually represents the conceptual design.

1.2.1 Entity Type Description

Entities describe real world-objects and are defined by their name and attributes they contain. For example in this in this model the most important entities are Employee, Client, Route. Now we will describe each entity name, attributes, domain, keys, candidate key.

Entity: Employee

Description: An Employee get paid by the landscaping company, also an Employee will be assign to a Car with two descriptive attributes start_date and end_date. Employee will work either as a driver-employee or just an employee, the distinction between them is that the one that drive earns more income.

Candidate key: employee_ID
Primary key: employee_ID
Strong/Weak Entity: Strong
Fields to be indexed: employee_Id, salary

Attribute name:	Employee_ID	Name	Address	Phone	Salary
Description	Unique id assigned to employees to track they everyday start - end work hours and their payment.	Name of the employee (First, Last)	Address of employee (123 Street St.)	Phone number to reach employee	The amount paid to each employee
Type	Integer	String	String	int	float
Value / Range	0-MaxID	any	any	000-000-0000 to 999-999-9999	0-9999
Null allowed	no	no	no	yes	yes
Unique	yes	no	no	yes	no
Simple / Composite	simple	composite	composite	simple	simple

Entity: Client

Description: A Client is most of the time the owner of the house on which the Landscaping company works on. Client may own many houses, Client may be a tenant. Client is also with whom the company signs the contract to start working on, and from whom the company gets its income.

Candidate key: client_ID
Primary key: client_ID
Strong/Weak Entity: Strong
Fields to be indexed: client_ID

Attribute name:	Client_ID	Name	Address	Phone
Description	Unique id assigned to clients to keep track of their houses and payments	Name of the client (First, Last)	Address of client (123 Street St.)	Phone number to reach client
Type	Integer	String	String	int
Value / Range	0-MaxID	any	any	000-000-0000 to 999-999-9999
Null allowed	no	no	no	yes
Unique	yes	no	no	yes
Simple / Composite Attributes	simple	composite	composite	simple
Single / Multi value	single	single	single	single

Entity: Income

Description: Income is a payment given to the company by the client in exchange for landscape services which might include extra services. Payment sometimes may be lower or higher than usual, for that we use description. Payments can be made via bank deposit, check or cash.

Candidate key: income_ID
Primary key: income_ID
Strong/Weak Entity: Strong

Fields to be indexed: income_ID, date, type, amount

Attribute name:	income_ID	date	type	amount	description
Description	Unique id assigned to each payment to keep track of them	- If cash, date the payment was received. - If check or money order, date on check. - If bank, date on payment received	Cash, check, money order, bank transaction	Amount each client pays for Landscape service	To record if there is something unusual on the amount or type of payment
Type	Integer	date	String	float	string
Value / Range	0-MaxID	date	any	any	any
Null allowed	no	no	no	no	yes
Unique	yes	no	no	no	no
Simple / Composite attribute	simple	simple	simple	simple	single
Single / Multi value	single	multi	single	single	single

Entity: Payment

Description: A payment is given to employee for his days working with the company. Payment can also be given as check, but mostly is cash.

Primary key: payment_ID
Strong/Weak Entity: Strong
Fields to be indexed: payment_ID, amount, date, type, hrs_worked

Attribute name:	payment_ID	date	type	amount	hrs_worked	description
Description	Unique id assigned to each payment to keep track of them	- If cash, date the payment was given. - If check or money order, date on check.	Cash, check,	Amount each employee gets paid	Total hours worked for the company per month	To record if there is something unusual on the amount or type of payment
Type	Integer	date	String	float	float	string
Value / Range	0-MaxID	date	any	any	0.0-MAXID	any
Null allowed	no	no	no	no	no	yes
Unique	yes	no	no	no	no	no
Simple / Composite attribute	simple	simple	simple	simple	simple	single
Single / Multi value	single	multi	single	single	single	single

Entity: Contract

Description: A contract is made for each client's house that the company will start their services with.

Primary key: contract_ID
Strong/Weak Entity: Strong
Fields to be indexed: contract_ID, date, price

Attribute name:	contract_ID	date	price	description
Description	Unique id assigned to each contract to keep track of them	Date the contract was signed	Amount client pays for service	Specification on obligations of landscape company on the client's house.
Type	Integer	date	float	string
Value / Range	0-MaxID	date	any	any
Null allowed	no	no	no	no
Unique	yes	no	no	no
Simple / Composite attribute	simple	simple	simple	single
Single / Multi value	single	multi	single	single

Entity: Tools

Description: this entity will store the tools the company uses to track which tools are taken in the truck in a specific date/route.

Primary key: tools_ID
Strong/Weak Entity: Strong
Fields to be indexed: tools_ID, brand

Attribute name:	tools_ID	brand	name	description
Description	Unique id assigned to each tool to keep track of them	Brand of the tool	Name of the tool	Specifications about the tools maintenance
Type	Integer	string	string	string
Value / Range	0-MaxID	any	any	any
Null allowed	no	no	no	yes
Unique	yes	no	no	no
Simple / Composite attribute	simple	simple	simple	single
Single / Multi value	single	single	single	single

Entity: Car

Description: Car is used by the employees to work on the routes, also to store the tools while working.

Primary key: car_ID
Strong/Weak Entity: Strong
Fields to be indexed: car_ID, start_date, end_date

Attribute name:	**car_ID**	**start_date**	**end_date**	**car_desc**
Description	Unique id assigned to each car to keep track of them	Date and hour car was started being in use	Date and hour car ended being in use	Specifications about the car maintenance
Type	Integer	date	date	string
Value / Range	0-MaxID	any	any	any
Null allowed	no	no	no	yes
Unique	yes	no	no	no
Simple / Composite attribute	simple	simple	simple	single
Single / Multi value	single	single	single	single

Entity: House

Description: A house is where the company does its landscape maintenance service every week.

Primary key: house_ID
Strong/Weak Entity: Strong
Fields to be indexed: house_ID, address, start_date, end_date

Attribute name:	house_ID	address	start_date	end_date
Description	Unique id assigned to each house to keep track of them	Address of house	Date and hour house service started	Date and hour house service ended
Type	Integer	string	date	date
Value / Range	0-MaxID	any	any	any
Null allowed	no	no	no	no
Unique	yes	no	no	no
Simple / Composite attribute	simple	composite	simple	simple
Single / Multi value	single	single	single	single

Entity: Route

Description: The houses employees have to do service in a specific day are called a Route. There can be different routes in a week. A unique route for every week day.

Primary key: route_ID
Strong/Weak Entity: Strong
Fields to be indexed: route_ID, start_date, end_date

Attribute name:	**route_ID**	**start_date**	**end_date**
Description	Unique id assigned to each route to keep track of them	Date and hour route was started	Date and hour route ended
Type	Integer	date	date
Value / Range	0-MaxID	any	any
Null allowed	no	no	no
Unique	yes	no	no
Simple / Composite attribute	simple	simple	simple
Single / Multi value	single	single	single

Entity: Additional_Service

Description: Additional services given to the client which were not written on the contract or added later, or just one time service.

Primary key: service_ID
Strong/Weak Entity: Strong
Fields to be indexed: service_ID, price

Attribute name:	**service_ID**	**date**	**price**	**description**
Description	Unique id assigned to each additional service to keep track of them	Date the additional service was done	Amount client paid for additional service	Specification on what done on the service
Type	Integer	date	float	string
Value / Range	0-MaxID	date	any	any
Null allowed	no	no	no	no
Unique	yes	no	no	no
Simple / Composite attribute	simple	simple	simple	single
Single / Multi value	single	multi	single	single

1.2.2 Relationship Type Description

Relationships between two or more entities exists when we associate them with a type. They can be defined by the entities they associate as well as attributes that help describe the relationship. They also distinguish constraints that control the amount of entities related to one another. Relationships can be described by the entities types they associate, the purpose of the relationship, the entity set involved, mapping cardinality, and any additional attributes or constraints that help identify the relationship.

Relationship: produced by
Description: Income is generated from the houses, clients, and services provided to consumers. We describe the relationship between income and contract as income produced by a contract entity.
Entity Sets Involved: Income, Contract
Mapping Cardinality: N .. 1
Descriptive Field: None
Participation Constraint: Income will have total participation while a contract will have partial participation.

Relationship: written for
Description: Every contract will be tied to a house. We describe this relationship between contract and house as contract written for a house.
Entity Sets Involved: Contract, House
Mapping Cardinality: 1 .. 1
Descriptive Field: None
Participation Constraint: A contract will have total participation and house will have total participation since every contract will be tied to a house.

Relationship: provided to
Description: Additional services can be provided to a house that may or may not be under contract. We describe this relationship between additional service and house as an additional service provided to a house.

Entity Sets Involved: Additional Service, House
Mapping Cardinality: N .. 1
Descriptive Field: None
Participation Constraint: An additional service will have partial participation while a house will have total participation if an additional service is being provided.

Relationship: owned by
Description: Every house is owned by someone and in our database we describe these owners as clients. Clients will be the ones being provided and charged for the services. We describe this relationship as a house is owned by a client.
Entity Sets Involved: House, Client
Mapping Cardinality: N .. 1
Descriptive Field: None
Participation Constraint: In this relationship the house has total participation since it depends on a client to own it. Client also is a total participation since every client owns a house.

Relationship: assigned
Description: Every house is assigned a route in order for the employees to keep track of the houses to service during the week. A route will consist of a list or schedule of houses. We describe this relationship as every house is assigned a route.
Entity Sets Involved: House, Route
Mapping Cardinality: 1 .. 1
Descriptive Field: None
Participation Constraint: A house in this relationship will have total participation since every house needs a route to be worked on by a route. A route has partial participation since not every route is tied to a house. A route can have different houses but not assigned to every house in the house set.

Relationship: worked by

Description: A route is worked by an individual car in the set. Every route will be different so we need to distinguish which car is working on which route. We describe this relationship as every route is worked by a car.

Entity Sets Involved: Route, Car

Mapping Cardinality: 1 .. 1

Descriptive Field: None

Participation Constraint: A route will have total participation since every route will have a car it is worked by in the set. A car will have partial participation since if a car changes routes or a car is out of maintenance not every car will have a route.

Relationship: contains

Description: A car will carry all the appropriate tools in order to perform the services required by the employees. A car will be loaded with tools that will be kept track of in order to have an accurate inventory. We describe this relationship as a car contains this set of tools.

Entity Sets Involved: Car, Tool

Mapping Cardinality: 1 .. N

Descriptive Field: None

Participation Constraint: Car in this relationship will have total participation since every car will need a set of tools. Tools will have total participation since every tool will be dependent of the car it is contained in.

Relationship: assign to

Description: A car will car a particular set of employees or an individual employee. The employees will ride in a particular car to go out and perform services required of them. We describe this relationship as a car will be assigned to an employee or set of employees.

Entity Sets Involved: Car, Employee

Mapping Cardinality: 1 .. N

Descriptive Field: Start_Date, End_Date

Participation Constraint: Car will have total participation since every car will be assigned to an employee if in use. Employee will have partial participation because not every employee in the database will necessarily ride in a car but can be assigned a car.

Relationship: paid
Description: Employees that will work for the business have to be paid for the work being provided. In order to keep track of payments and information on those payments to employees we have a payment database. We describe this relationship as an employee is paid a payment.
Entity Sets Involved: Employee, Payment
Mapping Cardinality: 1 .. N
Descriptive Field: None
Participation Constraint: Employee will have partial participation in that an employee is not dependent of a payment. A payment may not be given to an employee if a worker is not being paid for some reason. Payment will have total participation since it will be dependent of an employee and a payment will not be made if no employee existed for it.

1.2.3 Related Entity Type

Specialization is the process of defining subclasses of a specific entity type in which attributes of the superclass are transferred down to the subclass. The subclass then has its own distinguishing attributes that help define it and give reason for the subclass.

Generalization is the reverse process of specialization in that generalization is a bottom-up approach to combining classes into a superclass. and eliminates unnecessary redundancy. Tools in our database is an example of this process, since instead of breaking the tools down into subclasses of different types we generalized it to a tool entity type.

Aggregation is the process of relating two entities as a single entity or group. The purpose is to gather more information by specifying only a certain group or

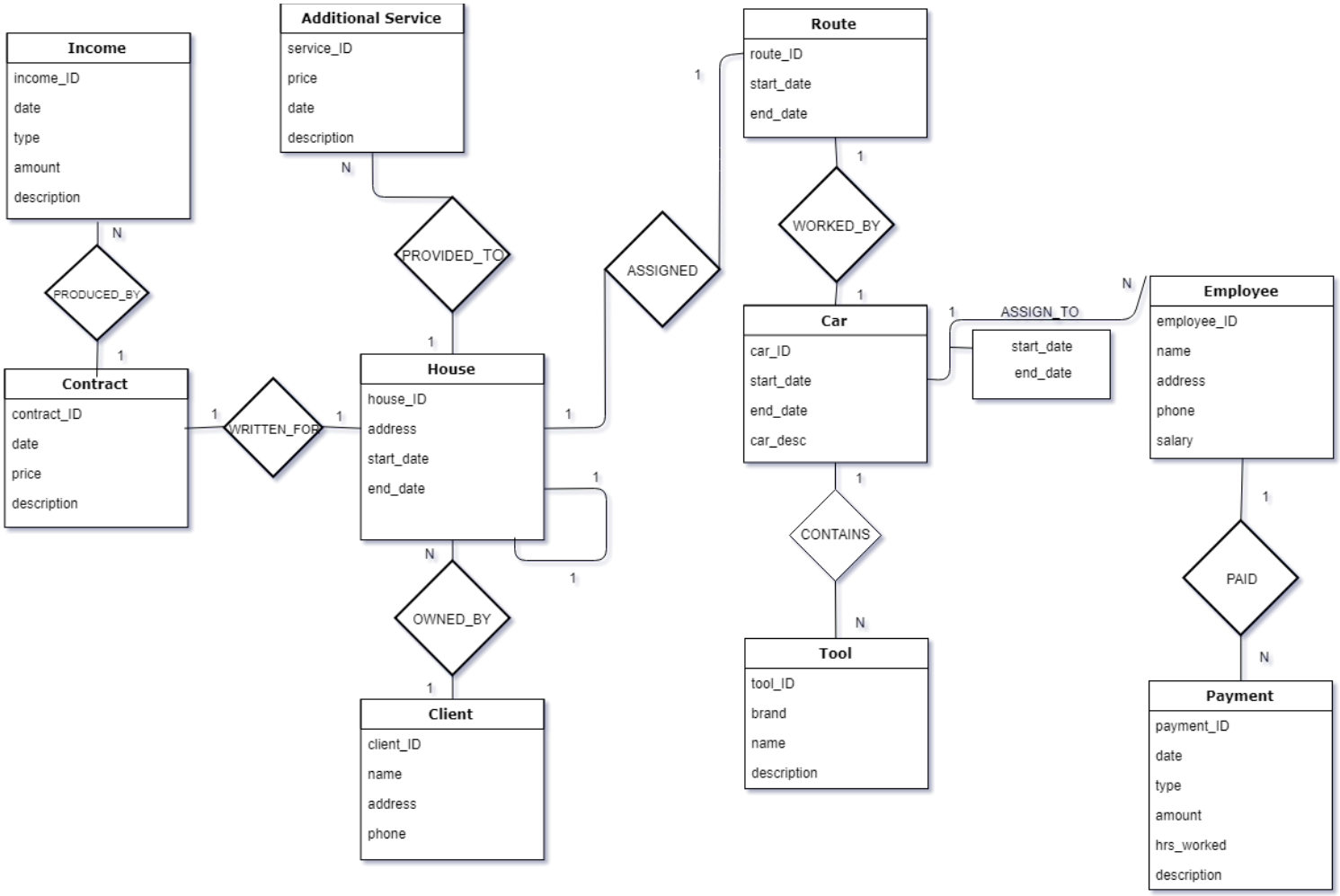
attribute from multiple entities. It creates a whole object instead of individual components.

Specializations and generalizations contain constraints of disjointedness and completeness. The disjointedness constraints specifies that a specialization subclasses are disjoint. This means the entity can only be a only one of the specialization classes. The completeness constraint can either be total or partial. Total completeness says that every entity in a superclass is at least member of one of the subclass. Partial completeness constraint specifies that an entity does not have to be a member of any subclass.

1.2.4 E-R Diagram

A helpful tool to visualize what is being produced using the ER model we make the use of a ER diagram. The ER diagram associates the entities and relationships of our model in a way that can be understood using visual objects and flow. The relationships in the diagram will be represented using lines and connected in between will be a relationship type represented by a diamond box with relationship type name. Entities will be represented using a square box with entity name and attributes written inside the box. Included with the relationships and entities will be symbols “1” or “N” to represent the cardinality of the relationship.

The ER diagram for Green Landscaping is as follows:


Income
income_ID
date
type
amount
description
Additional Service
service_ID
price
date
description
Route
route_ID
start_date
end_date
PRODUCED_BY
PROVIDED_TO
ASSIGNED
WORKED_BY
Contract
contract_ID
date
price
description
WRITTEN_FOR
House
house_ID
address
start_date
end_date
Car
car_ID
start_date
end_date
car_desc
ASSIGN_TO
start_date
end_date
Employee
employee_ID
name
address
phone
salary
CONTAINS
PAID
OWNED_BY
Client
client_ID
name
address
phone
Tool
tool_ID
brand
name
description
Payment
payment_ID
date
type
amount
hrs_worked
description

PHASE 2: Conceptual Database and Logical Database

A conceptual database is a visual representation of a database meant to make it easier to comprehend as a concept. On the other hand, a logical database is a software based representation on how data will be represented and stored. The overall notion for this phase is to process how an ER Model is converted to a relational model and give a thorough description of the relational model, its structure, queries, and expressions.

In each section, the development process of a conceptual database and logical database will be described. Beginning in section 2.1, we will discuss the importance, meaning, and difference between E-R model and relational model. Next in section 2.2, we will describe the conversion from conceptual database model to a logical database model. After that in section 2.3, we will convert the Green Landscape database into a relational database. In section 2.4, we will demonstrate sample queries for the Green Landscape database as well as relational, tuple, and domain calculus expressions.

2.1 E-R model and Relational Model

The ER Model is categorized as a conceptual database design while the relational model is described as a design implementation of a logical database. In this section we will discuss both models and compare how they are different.

2.1.1 Description of E-R model and Relational Model

The ER Model was developed by Peter Chen in 1976 for his paper “The Entity-Relationship Model: Toward a Unified View of Data”. In this paper he proposed the model, describes its semantics, and provides an example of how to design a database. As the title of the paper states, Peter Chen called it a way to unify data and visually represent it. The purpose was to make it easier for a developer to present their ideas, data, and relationships into one unified model. The model then can be interpreted by both industry experts and non-experts.

The relational model was developed by Edgar F. Codd in 1969 for his research report “Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks”. In this report he theorized the complexity and large amount of data that would be stored in the future. He wanted to created a way to logically view the data and detect any inconsistencies in the data bank. The model is

therefore organized into columns and rows with a table, or relation. Each table represents an entity type while the rows represent the instances of that entity type. Columns represent the values of the attributes of that instance within the relation. This model is the foundation to most modern database systems such as PostgreSQL and MySQL. The relational model also devolves into the formal query languages of relational algebra, relational calculus, and domain calculus.

The ER model and relational model are distinguished by the way the data is visualized in the model. The ER model focuses on representing physical or conceptual objects as entities and the interaction or association between entities as relationships. The model is usually visualized as a diagram with boxes representing the entities, lines representing the flow, and either diamonds or boxes to represent the relationships. Entities have added attributes that describe the entity in terms of values that will be important to define the entity within the database. Relationships can also have attributes but are not necessary to the model.

The relational model focuses on representing the data that will be inputted within the database. The data, entities, and relationships are all represented with the table of the model. The main features of the model are to represent the relation as a table with columns representing the attributes, and rows representing the tuple of information. The tuple is the instance of the entity and a set of tuples is called a record.

The purpose of each model is to help visualize and organize data that will either be implemented into a database or already represents a database. The ER model's purpose is to help visualize the structure of how entities relate with each other. The intended concept being that by visualizing it in that way it will be easier to understand each entity and its importance within the database. On the other hand, the relational model's intended purpose was to help visualize how a programmer can implement the data into a database. The tables help visualize data going directly into your database by the use of columns and rows like any other table of data. This model also correlates more with formal query languages therefore makes it easier to translate into SQL languages.

2.1.2 Comparison of E-R model and Relational Model

The ER model and Relational model both represent the same data but each model structures it differently. The ER model makes it easier for a person that is not tech savvy or not a programmer to visualize the data and understand the concept of the database. This can be useful when a customer or manager is not knowledgeable of database system but cooperation between the two parties is

needed to create a database. Another advantage of the ER model is alterations to the model can be done with ease since design specifics are not a part of the model. If the customer is not satisfied or makes changes to how they want the database then it does not affect the implementation. Scalability before the database is coded is done with no problem. The disadvantage of the ER model is that it is not based on mathematical languages and instead is just a visual representation. The model is abstract and does not represent a standard for implementing a database. The developers who will actually code the database are left with an ambiguous model that can be interpreted differently by different people.

The major advantage of the relational model is it is developer friendly and gives an accurate representation of how the data will be instantiated within the database. The relational model is based on formal mathematical query languages therefore its translation into SQL and programming languages is very simple. Tables are an advantage to developers since it gives a visual representation of how the data is setup. The disadvantage of the relational model is it depends on a level of comprehension of database software to be able to understand it. A person that lacks knowledge of software or mathematical languages will have a difficult time interacting with the model and understanding its features.

2.2 Conversion from Conceptual Database Model to Logical Database Model

After defining the ER model and relational model, in this section we will focus on how to convert from the ER model into the relational model. In order to accomplish this we must give a detailed method from ER specific components to relational model components. The first conversion will involve how entity types are converted into relations in section 2.2.1. Secondly we must detail the process of converting relationship types to relations in section 2.2.3. Finally, in section 2.2.3 we give the constraint details of the database to be implemented.

2.2.1 Converting Entity Types to Relations

In order to begin the conversion from the ER model to the relational model, it is necessary to directly convert entity types to relations. Entity types are represented as a set of relation schemas in a relational model. Each relation scheme is made up of a list of attributes with single-value domains. In the ER model the entity types are more complex, they are composed of composite or multi-value attributes as well as the possibility of it being a weak entity type with no key.

We will now discuss how to represent both strong and weak entity types as relations in this section. We will then explain how simple and composite entity

attributes can be represented as relation attributes with atomic domains. Atomic domains being those that cannot be broken down and represented into any simpler values. We will then demonstrate how multi-value and single value entity attributes are represented as relation attributes.

Strong Entities to Relations

For strong entity types to be converted we must show that each entity (E) is converted into one relation schema (R). The schema will be a direct conversion from strong entity to relation so it will have the same name. The single-value and simple attributes related to the strong entity type are then transferred to the relation that mapped to it. From the entity in the ER model, a unique key is chosen to be primary key attribute of the relation schema. Any additional keys then become candidate keys of the relation schema. While candidate keys are also unique, they will not be used to represent any tuples or rows within the relation since a primary key was established.

Weak Entities to Relations

For weak entity types, the process is similar to the strong entity types in that it can be directly represented a relation schema of the same name. Weak entity types then have to transfer over a foreign key to represent the primary key in a relation. If the foreign key is a key to another weak entity then a strong entity must be formed in order for the process to be correct. The primary key can be made by using key attributes along with partial key attributes that form a strong entity. All the rest of the partial keys and attributes of the weak entity will then also be converted over to the relation schema. The importance here being that any weak entities must be converted into strong entity types in order to be represented and converted into a proper relation.

Simple and Composite Attributes

Simple and composite attributes differentiate in the way that the must be converted into the relational schema. The simple attributes directly convert over to the relation schema R that corresponds to the entity type. The way composite attributes convert over is they first break down into simple attributes. After they are broken down, they convert over as simple attributes into the relation schema R.

Single and Multi-value Attributes

Single and multivalued attributes also differentiate in the way they are processed and converted over from entity type to relation schema. The single-valued attributes are easy to transfer over, similar to the process of

converting the simple attributes. The attributes are added over to the relation schema that is corresponds to the entity type. For multi-value attributes the process is a little more complex. The attributes must be represented by a separate relation R_A. The primary key for this new relation is made up by the multi-value attributes combined with the foreign key corresponding from the entity type involved. If the multi-value attribute is composite then only the unique simple components become part of the relation R_A's primary key.

2.2.2 Converting Relationship Types to Relations

The conversion of relationship types to relations can be more complex since an ER model focuses on both entities and relationships while the relational model only focuses on relations. Representing relations within a relation schema then becomes a process of separating relationships into new relations or adding attributes into existing relations. In this section, we will discuss how to represent relationships types in the following situations:

- Relationship types with 1:1, 1:N, M:N
- "IsA" and "HasA" concepts within superclasses and subclasses
- Relationship type involving other relationship type
- Recursive relationship
- Relationships involving more than two entity types
- Relationship and Categories (or union) types

Relationship Types with 1:1 Cardinality Constraint

An entity relationship between entity A and entity B with 1:1 cardinality should convert into a mapping of relation R_A and relation R_B. Every instance in R_A should relate to every instance in R_B with the relational model. The constraint is then represented by one of three methods:

- **Foreign Key Approach**: Here, the primary key within the first relation is made into a foreign key attribute of the second relation. All the attributes that correspond with the relationship type containing the primary become additional attributes of the relation that now holds the foreign key.
- **Merged Relation Approach:** Here, the attributes of both relations are combined and made into one relation within the relational model.
- **Cross Reference Approach:** In this approach, the relationship type in the ER model becomes a new relation within the relational model. The primary keys from both relation R_A and R_B will be used as foreign keys within the new relation. The simple attributes along with the simple components from the

composite attributes will also be transferred into the new relation. The primary key of the new relation will be one of the foreign keys acquired.

Each of these approaches can have benefits over the other as well as its disadvantages. The foreign key approach has the benefits of creating a unified approach to interacting with the relations making up the relationship. The disadvantage being the approach only works if both entity types contain total participation. Cross reference approach is good if neither entity has total participation but creates more joins when performing any queries on the relations.

Relationship Types with 1:N Cardinality Constraint

Following the previous model, a relationship between entities A and B with cardinality 1:N can be mapped with the conversion into relations R_A and R_B. The difference now being every instance within R_A will map to multiple instance of R_B. Every instance within R_B will map back to only one instance within R_A. The constraint is then represented with one of the two following methods:

- **Foreign Key Approach:** This approach involves a similar approach as the foreign key approach with converting 1:1 cardinality type. The difference being the foreign key and relationship type attributes converted into the relation will be derived from the entity with "N" of the relationship. Entities on the "N" side of the relationship can only be related to one instance of the entity type "1" side.
- **Cross Reference Approach:** Same as the 1:1 relationship type but the primary key of the new relation will be derived from the foreign key that belongs to the relation converted from the "N" side of the relationship.

The comparison between the two approaches is similar to the approaches used for the 1:1 cardinality constraint relationship conversions.

Relationship Types with M:N Cardinality Constraint

The method of conversion for this relationship type constraint is the cross reference or relationship relation approach. Again, this involves creating a new relation for every relationship type. The foreign key attributes in the new relation are the primary keys from the entity types involved in the relationships. The combination of the foreign keys will create the primary key for the relation. Any simple attributes of the relationship type will become attributes of the new relation.

Superclasses and Subclasses for the "IsA" Relationship

The “IsA” relationship type happens when entity types can be classified as disjoint subclasses of a superclass entity. This means that an entity is made up of no more than one subclass. To represent this relationship type in the relational model there are three methods:

- **Multiple relations - superclass and subclass**: With this approach, a superclass entity becomes a relation R_{super} and subclass entity becomes a relation R_{sub}. The superclass relation is composed of the attributes from the superclass entity. The subclass relation is composed of the attributes of the subclass entity as well as a foreign key made from the primary key of the superclass relation. The foreign key also serves as the primary key of the relation.
- **Multiple relations - subclass only**: With this approach, the subclass entities are converted into their own relations. The subclass relation then is composed of the attributes from the subclass entity and superclass entity types.
- **Single relation with one type attribute** - With this approach, one relation is created that contains the union of the attributes from the superclass and from all of the subclasses. The relation also contains a type which distinguishes which of the subclasses this new relation belongs to.

Advantages for the first method include it being useful to work for most superclass and subclass relationship types. The second method is good because it means there will be fewer operations needed between relations. The third method has the least amount of operations needed making it extremely efficient if the entities align properly.

Disadvantages for the first method include it requires the most operations to combine the relations. The second method has the disadvantage of only working with relationships were the participation is total. The third method is not optimal because it creates a very large relation and it is only useful if the subclasses being unioned together contain attributes that are similar or equal to each other.

Superclasses and Subclasses for the “HasA” Relationship

The “HasA” relationship happens when a superclass entity type is composed of subclass entity types that are overlapping. The superclass then belongs to multiple subclasses. To represent this relationship type in a relational model there are two methods:

- **Multiple relations - superclass and subclass**: The approach here is the same as the “IsA” relationship. The superclass and subclass entity types become their own relations respectively. The attributes follow the same mapping as well.
- **Single relation with multiple type attributes** - With this approach, one relation is made from the union of attributes from the superclass and all of the subclasses combined. The relation then contains a boolean attributes for every subclass belonging to the relationship types. A “true” value signifies that the tuple belongs to that subclass.

The advantages and disadvantages for the first method are exactly the same as the “IsA” method using multiple relations. For the second method the advantage is it requires less joins. The disadvantage for this method is that the attributes for the subclasses that a relation isn’t belonging to makes it NULL. Thus, space is wasted by containing these attributes within certain relations.

Relationships involving other Relationship Types

Relationships involving other relationship types are mapped by using a single primary key attribute for the relationship type. The relationship is then mapped into the relational model using the foreign key approach or cross reference approach based on the cardinality of the relationship. The primary key within the relationship type will be the base for the foreign keys in the methods.

Recursive Relationships

Recursive relationships happen when an entity type in the ER model is related to itself. To represent this relationship type within the relational model there are two methods:

- Foreign Key Approach: With this approach we map the relationship into a relation. The relation then contains a foreign key attribute that is composed of the primary key of the same relation.
- Cross Reference Approach: With this approach, a new relation for the recursive relationship is made. Then it contains two foreign keys that reference the primary key of the original relation. The combination of the foreign keys forms the primary key of the new recursive relation.

Both methods contain advantages and disadvantages depending on the layout of the types and attributes. The advantage of foreign key is it makes for less join operations. The advantage for cross reference approach is it eliminates the NULL values that are not participating in the relationship. The disadvantage for the foreign key approach is that it includes tuples that are NULL but required to

represent the relation. The only disadvantage of the cross reference is it requires more join operations when making queries.

Relationships involving more than Two Entity Types

For relationships involving more than two entity types, a new relation is created to represent the relationship type. Every entity type composing the relationship is automatically already converted into their own relations. The new relation then contains foreign key attributes made up from the primary keys of each of the entity types. The combination of all the foreign keys then makes up the primary key for the new relation. Any foreign key created from the "1" side of an 1:N cardinality are excluded since they don't have the necessary participation.

Relationship and Categories (or Union) Types

For categories or union relationship types, multiple superclass entities relate to a subclass entity. The relations corresponding to different superclass entities are already given different primary keys when converted over. To represent the union relationship, the superclass relations have surrogate key attributes added into them. When multiple entities are superclasses of a subclass entity the relation tuples corresponding to the superclasses share the same surrogate key.

2.2.3 Database Constraints

Constraints are placed within a database to make sure that all of the data that is stored is significant and useful. The tuples (rows) within the relation therefore has to satisfy certain conditions and rules which are constraints that allow the database to be represented correctly. If any of these constraints are violated then the data becomes meaningless and therefore the relation state is now invalid. In order to understand constraints within a database management system it will be useful to define the following constraints:

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

Domain Constraints

The domain constraints are placed in order to ensure that values within the tuples of a relation state correspond with the attribute's domain in the schema. This includes restrictions on the values to specify the data type to be given to that attribute. The database management system is in charge of rejecting any INSERT or

UPDATE operations that would cause tuples to have invalid values. It can also make invalid values NULL or “default” depending on the constraints involved.

Entity Constraints

Entity constraints focus on verifying that all the tuples of a relation state have a valid primary key and that a NULL value is not present within its state. If the constraint is violated it is impossible to accurately or uniquely identify the tuple making the database next to useless when querying.The database management system enforces any INSERT or UPDATE operations and rejects any primary key attributes set to NULL.

Primary Key and Unique Key Constraints

Primary Key and Unique Key Constraints focus on providing tuples with a uniquely identifiable state within the relation or table. To ensure this, tuples must include a minimal set of attributes, or a key, whose values are never the same for any two tuples in the relation state. The primary key constraint therefore guarantees that no two tuples will have the same value for the primary key attribute. Attributes within a relation state can also have a uniqueness constraint if they are not the primary key. Within either case, the database management system can reject any INSERT or UPDATE operations that contain invalid values or existing values in the primary key attribute. The database system can also enforce a constraint by making the primary key value auto-increment during the INSERT operation of new tuples.

Referential Constraints

Relation schema contain foreign keys in the form of attributes therefore a tuple within the relation state which the foreign key references must exist in order for the relation schema to be valid. The referential constraints prevents tuples from being created that have foreign keys belonging to another tuple. The two tuples being distinct tuples in the relation but not containing the same primary key.

The referential constraint must be enforced by the database system for the INSERT, UPDATE, and DELETE operations. The INSERT operation rejects any new tuple that has a foreign key value that is either invalid, or assigns the foreign key to be NULL or default under certain conditions. The DELETE operation has three possible options:

- **Restrict:** With the restrict option an operation that attempted to delete a tuple referenced by a foreign key of another tuple will be rejected.
- **Cascade**: With this option the database system will delete all the tuples that reference the deleted tuple through the foreign key.
- **Set Default**: With this option the tuple will be deleted and any foreign key values referenced to it will be set to NULL or default.

The UPDATE operation is rejected if the foreign key value becomes invalid or preceding options can be followed to allow the operation to proceed.

Check Constraints and Business Rules

All of the constraints and rules above are included with a database management system, but a business might decide to implement additional rules specific to their system. The business will then develop additional rules coded into the application using the database that then will need to be followed by designers. It is important to know these new constraints and rules specific to the business.

2.3. Convert Green Landscape Conceptual Database into a Relational Database

In this section we convert all of the entity types and relationship types into relational schema for the Green landscape. Constraints will be placed on the entity and relationship attributes for the sake of organization and logic. Sample tuples for each relation show how the relationship would look in the real world.

2.3.1 Relation Schema

All of the Green Landscape's relationship schemas are listed below. This includes constraints for all entity relationship type attributes.

Relation Schema: Employee

employee(**employee_ID, payment_ID,** date, type, amount, hrs_worked, description)

Attributes

*employee_ID	Integer, 1-1000, primary key
*paymentID	Integer, 1-1000, foreign key
name	varchar(50)
address	varchar(100)
phone	Integer, 0000000000 - 0000000000
salary	decimal (7,2)

Candidate Keys: employee_ID (primary)

Primary Key/Entity Integrity Constraint: “employee_ID” must be unique and cannot be NULL

Uniqueness Constraint: “employee_ID” and **“**phone” must be unique

Derivation From Entity Relationship Types: car

Derived from car entity type. Employee “assigned_to” car.

Relation Schema: Payment

payment(**payment_ID,** date, type, amount, hrs_worked, description)

Attributes

*payment_ID	Integer, 1-1000, primary key
date	date
type	varchar(20)
amount	decimal (7,2)
hrs_worked	decimal(2,1)
description	varchar(255)

Candidate Keys: payment_ID(primary)

Primary Key/Entity Integrity Constraint: “payment_ID” must be unique and cannot be NULL

Uniqueness Constraint: “payment_ID” must be unique

Derivation From Entity Relationship Types: employee

Derived from employee entity type. Payment “paid” employee.

Relation Schema: Route

route(**route_ID, car_ID,** start_date, end_date)

Attributes

*route_ID	Integer, 1-1000, primary key
*car_ID	Integer, 1-1000, foreign key
start_date	date
end_date	date

Candidate Keys: route_ID(primary)

Primary Key/Entity Integrity Constraint: “route_ID” must be unique and cannot be NULL

Uniqueness Constraint: “route_ID” must be unique

Derivation From Entity Relationship Types: route

Relation Schema: Car

car(**car_ID,** start_date, end_date, car_desc)

Attributes

*car_ID	Integer, 1-1000, primary key
*tool_ID	Integer, 1-1000, foreign key
start_date	date
end_date	date
car_desc	varchar(255)

Candidate Keys: car_ID(primary)

Primary Key/Entity Integrity Constraint: “car_ID” must be unique and cannot be NULL

Uniqueness Constraint: “car_ID” must be unique

Derivation From Entity Relationship Types: route

Derived from route entity type. Route “worked_by” car.

Relation Schema: assign_to

assign_to(**car_ID, employee_ID,** start_date, end_date)

Attributes

*car_ID	Integer, 1-1000, primary key
*employee_ID	Integer, 1-1000, primary key
start_date	date
end_date	date

Candidate Keys: [car_ID, employee_ID]

Primary Key/Entity Integrity Constraint: "car_ID" must match the car's "car_ID", "employee_ID" must match the employee's "employee_ID".

Derivation From Entity Relationship Types: Car *assign_to* employee

Derived from using the cross-reference; 1..N relationship approach between Car and Employee.

Relation Schema: Tool

tool(**tool_ID, repair_ID,** brand, name, description)

Attributes

*tool_ID	Integer, 1-1000, primary key
*repair_ID	Integer, 1-1000, primary key
brand	varchar(50)
name	varchar(50)
car_desc	varchar(255)

Candidate Keys: tool_ID(primary)

Primary Key/Entity Integrity Constraint: “tool_ID” must be unique and cannot be NULL.

Uniqueness Constraint: “tool_ID” must be unique

Derivation From Entity Relationship Types: car

Derived from car entity type. car “contains” tool.

Relation Schema: Repair

repair(**repair_ID,** supplier, date, cost, description)

Attributes

*repair_ID	Integer, 1-1000, primary key
supplier	varchar(100)
date	date
cost	decimal(7,2)
description	varchar(255)

Candidate Keys: repair_ID(primary)

Primary Key/Entity Integrity Constraint: “repair_ID” must be unique and cannot be NULL.

Uniqueness Constraint: “repair_ID” must be unique

Derivation From Entity Relationship Types: tool

Derived from tool entity type. tool “has” repair.

Relation Schema: House

house(**house_ID, service_ID, client_ID, contract_ID,** address, house_number, street_name, city, zip_code)

Attributes

*house_ID	Integer, 1-1000, primary key
*service_ID	Integer, 1-1000, foreign key
*client_ID	Integer, 1-1000, foreign key
*contract_ID	Integer, 1-1000, foreign key
number	varchar(7)
street_name	varchar(30)
city	varchar(30)
zip_code	varchar(9)

Candidate Keys: house_ID(primary)

Primary Key/Entity Integrity Constraint: “house_ID” must be unique and cannot be NULL, “contract_ID” must be unique and cannot be null

Uniqueness Constraint: “house_ID, ” “contract_ID” must be unique

Derivation From Entity Relationship Types: house

Relation Schema: Additional_Service

additional_service(**service_ID,** price, date, description)

Attributes

*service_ID	Integer, 1-1000, primary key
price	decimal(7,2)
date	date
description	varchar(255)

Candidate Keys: service_ID(primary)

Primary Key/Entity Integrity Constraint: “service_ID” must be unique and cannot be NULL.

Uniqueness Constraint: “service_ID” must be unique

Derivation From Entity Relationship Types: house

Derived from house entity type. additional_service “provided_to” house.

Relation Schema: Client

client(**client_ID,** name, address, phone)

Attributes

*client_ID	Integer, 1-1000, primary key
name	varchar(30)
address	varchar(150)
phone	varchar(10)

Candidate Keys: client_ID(primary)

Primary Key/Entity Integrity Constraint: “client_ID” must be unique and cannot be NULL.

Uniqueness Constraint: “client_ID” must be unique, “phone” must be unique

Derivation From Entity Relationship Types: house

Derived from house entity type. House “owned by” client.

Relation Schema: Contract

contract(**contract_ID, income_ID,** date, montly_fee, description, sDate, eDate)

Attributes

*contract_ID	Integer, 1-1000, primary key
*income_ID	Integer, 1-1000, primary key
montly_fee	decimal(7,2)
description	varchar(250)
sDate	date
eDate	date

Candidate Keys: contract_ID(primary)

Primary Key/Entity Integrity Constraint: “contract_ID” must be unique and cannot be NULL.

Uniqueness Constraint: “contract_ID” must be unique

Derivation From Entity Relationship Types: house

Derived from house entity type. Contract “written_for” house.

Relation Schema: Income

income(**income_ID,** date, type, amount, description)

Attributes

*income_ID	Integer, 1-1000, primary key
date	date
type	varchar(150)
amount	decimal(7,2)
description	varchar(250)

Candidate Keys: income_ID(primary)

Primary Key/Entity Integrity Constraint: “income_ID” must be unique and cannot be NULL.

Uniqueness Constraint: “income_ID” must be unique

Derivation From Entity Relationship Types: contract

Derived from contract entity type. Income “produced_by” contract.

2.3.2. Sample Data of Relation

Below is a list of tuple data for the Green Landscape's schema. This data represents the real-world scenario for all entities and relationships types. The tuples will be listed in a table format, were relational schema attributes are columns and individual tuples are rows. Each relation derived from another entity will have 1 to 10 sample tuples, while others will have 20 samples

Employee					
employee_ID	**payment_ID**	**name**	**address**	**phone**	**salary**
1	9	Eadie Paulus	734 Hollow Ridge Parkway	500-707-0171	$67.67
2	1	Annabal Brosio	5 Nova Crossing	407-885-0736	$105.72
3	8	Brit Tumilson	717 6th Street	354-433-9490	$81.68
4	2	Lonna McCutcheon	57 Jenna Trail	440-225-1045	$85.08
5	18	Evered Mizen	55 Sage Hill	951-672-7627	$68.62
6	3	Eldridge Coundley	19 Bayside Court	299-920-5922	$90.76
7	10	Gelya Mingotti	280 Tennyson Pass	991-246-3112	$119.87
8	20	Robers Decruse	2 Macpherson Way	938-190-3490	$92.05
9	4	Leese Ferris	0991 Ludington Street	204-527-6807	$77.15
10	19	Auberta Richly	023 Everett Parkway	871-947-1668	$76.83
11	6	Laverne Attwell	5 Northview Road	406-540-8813	$60.64
12	11	Saundra Villiers	17132 Dahle Avenue	718-994-0342	$74.34
13	12	Alistair Iveson	4 Weeping Birch Pass	748-461-1464	$69.54
14	5	Evy Heimann	59668 Merchant Court	946-249-0230	$85.35
15	13	Clarine Thews	665 Redwing Pass	474-581-2039	$113.67
16	15	Marleen Chaffen	092 Utah Lane	513-765-7735	$118.79

17	7	Osbourne Foort	8 Logan Junction	385-397-5671	$108.48
18	16	Atlante Gonoude	23 Mccormick Pass	519-601-7836	$91.52
19	14	Hazlett Summerrell	5025 Riverside Alley	915-749-7550	$89.59
20	17	Danette Mowlam	04 Kenwood Pass	611-133-8804	$118.39

Payment					
payment_ID	**date**	**type**	**amount**	**hrs_worked**	**description**
1	5/4/2018	cash	$109.90	5	Maecenas pulvinar lobortis est.
2	3/3/2018	cash	$80.01	8	Maecenas tristique, est et tempus semper, est quam pharetra magna, ac consequat metus sapien ut nunc.
3	7/19/2018	bank deposit	$102.46	6	Vestibulum rutrum rutrum neque.
4	12/7/2017	money order	$94.50	8	Duis consequat dui nec nisi volutpat eleifend.
5	6/19/2018	money order	$63.74	9	Phasellus sit amet erat.
6	6/12/2018	money order	$63.70	3	Aenean auctor gravida sem.
7	12/4/2017	cash	$78.01	10	Morbi non lectus.
8	3/13/2018	cash	$76.62	2	In blandit ultrices enim.
9	9/1/2018	cash	$78.01	2	Integer ac neque.
10	5/21/2018	cash	$87.06	6	Nam dui.

Repair				
service_ID	**supplier**	**date**	**cost**	**description**
1	Realfire	1/14/2018	$466.14	Morbi sem mauris, laoreet ut, rhoncus aliquet, pulvinar sed, nisl.
2	Brainsphere	11/26/2017	$536.68	Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.
3	Oyoyo	11/5/2017	$59.02	Aenean fermentum.
4	Zava	4/14/2018	$951.20	Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Mauris viverra diam vitae quam.
5	Yabox	11/23/2017	$312.79	Morbi a ipsum.
6	Gigazoom	9/13/2018	$749.54	Cras in purus eu magna vulputate luctus.
7	Eidel	3/17/2018	$862.23	Morbi a ipsum.
8	Topiczoom	10/4/2018	$381.30	Etiam vel augue.
9	Roombo	10/31/2017	$746.27	Sed sagittis.
10	Skimia	1/24/2018	$358.26	Nam tristique tortor eu pede.

Tool					
tool_ID	**brand**	**name**	**serviceID**	**car_ID**	**description**
1	Echo	Brush Cutter	9	3	Quisque ut erat.
2	Echo	Stump Grinder	2	2	Nulla ut erat id mauris vulputate elementum.
3	Honda	Blower	8	3	Morbi a ipsum.
4	Shindaiwa	Pole Pruner	5	1	Donec odio justo, sollicitudin ut, suscipit a, feugiat et, eros.
5	Shindaiwa	Tiller	9	2	Praesent blandit.
6	Honda	Blower	9	2	Nulla tempus.
7	Echo	Blower	5	4	Morbi non lectus.

8	Toro	Trimmer	6	7	Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.
9	Toro	Shredder	1	5	Phasellus in felis.
10	Blue-bird	Lawnmower	8	1	Morbi a ipsum.

Car			
car_ID	**start_Date**	**end_Date**	**description**
1	8/30/2018	4/12/2019	19XFB4F27FE859591
2	4/1/2018	5/29/2019	1G6DX67D170548661
3	11/23/2017	9/17/2019	1G4GA5GR9FF255678
4	2/1/2018	1/26/2019	WBAVC93588K828898
5	7/13/2018	8/16/2019	1FTMF1EW6AF048049

Assign_to		
car_ID	**employee_ID**	**date**
2	8	7/30/2019
3	3	11/15/2018
1	20	6/5/2019
3	18	10/5/2019
3	3	2/27/2019
2	2	6/9/2019
4	16	12/24/2018
3	13	4/3/2019
3	8	12/17/2018
3	12	2/27/2019

Route					
route_ID	weekday	sDate	eDate	house_ID	carID
1	Thursday	9/15/2019	7/24/2019	117	4
2	Wednesday	3/22/2019	8/10/2019	257	1
3	Wednesday	2/19/2019	4/21/2019	173	5
4	Wednesday	1/21/2019	1/2/2019	126	1
5	Tuesday	3/13/2019	7/15/2019	85	2
6	Monday	7/17/2019	6/17/2019	252	5
7	Tuesday	2/8/2019	10/19/2019	217	2
8	Friday	4/8/2019	1/26/2019	205	2
9	Thursday	3/12/2019	8/10/2019	212	3
10	Thursday	11/27/2018	11/18/2018	130	3
11	Wednesday	1/22/2019	3/16/2019	138	4
12	Wednesday	6/8/2019	5/6/2019	199	2
13	Monday	3/8/2019	8/7/2019	72	1
14	Saturday	4/6/2019	4/10/2019	2	2
15	Friday	3/6/2019	2/27/2019	13	5
16	Tuesday	11/28/2018	5/16/2019	93	2
17	Wednesday	9/26/2019	3/9/2019	199	2
18	Saturday	8/24/2019	2/4/2019	33	4
19	Wednesday	8/30/2019	5/11/2019	261	5
20	Wednesday	8/22/2019	7/16/2019	247	2

House								
house_ID	Number	Street name	zipCode	sDate	eDate	n_house_ID	contract_ID	client_ID
1	8	Manufacturers	75044	10/21/2019	10/6/2019	177	261	51
2	56734	Lerdahl		4/6/2019	11/14/2018	248	58	133
3	67	Rusk		4/14/2019	4/25/2019	42	24	187
4	4	Jenna	B3T	3/9/2019	9/10/2019	54	33	199
5	47645	Rowland	2965-212	9/17/2019	1/8/2019	253	108	38
6	0	Waubesa	8424	1/9/2019	11/18/2018	148	68	152
7	845	Nobel		4/30/2019	9/6/2019	173	66	167
8	43	Montana	673377	9/22/2019	5/12/2019	195	31	148
9	8	Gina	4216	4/24/2019	10/27/2018	166	241	123
10	14764	Bartelt		8/20/2019	5/6/2019	103	45	83
11	08	Eastwood	94180	12/29/2018	7/28/2019	84	202	54
12	8	Sutteridge	14604	10/5/2019	1/17/2019	60	266	188
13	179	Grayhawk		3/9/2019	5/16/2019	11	25	92
14	882	Novick		2/27/2019	2/10/2019	97	16	49
15	68	Orin	905 92	5/19/2019	8/4/2019	169	253	85
16	8	Fairview		2/23/2019	11/20/2018	220	53	158
17	8992	Village Green	0620	9/14/2019	5/4/2019	70	5	142
18	88380	American Ash		12/3/2018	3/6/2019	260	12	45
19	64	Esker		11/25/2018	9/12/2019	201	232	16
20	90	Mayer	72-510	2/8/2019	9/14/2019	108	122	103

Client			
client_ID	name	address	phone
1	Bernardina Connick	63 Nova Court	345-299-6113
2	Mabel O'Brogan	198 Lawn Road	959-718-3623
3	Eddie Noddings	555 Hayes Circle	733-226-2777
4	Darb Woloschinski	919 Lotheville Crossing	802-396-7730
5	Caty Charrette	0 Wayridge Plaza	788-538-9938
6	Linn Durnall	7599 Lerdahl Crossing	
7	Wilbur Craighall	68029 Carpenter Court	991-442-7082
8	Ailyn McArt	537 Portage Point	278-836-2460
9	Tonya Spread	23181 Lakewood Avenue	814-322-0320
10	Fancie Daybell	2 Merchant Road	793-398-6830
11	Bernete Demare	61775 Homewood Terrace	722-170-3517
12	Jess Ingree	6810 Blue Bill Park Drive	773-314-9708
13	Florri Bottleson	04 Randy Lane	393-145-9648
14	Pren Ianne	5394 Jay Place	612-754-2045
15	Calv Pellissier	30 Moland Point	336-562-0786
16	Mario Verbeek	3 Schlimgen Crossing	473-550-9225
17	Sergeant Silman	9328 Spenser Way	567-580-7840
18	Cesar Heersema	7 Havey Terrace	923-574-3611
19	Cy Bourrel	4 Steensland Hill	664-154-2943
20	Albert Muspratt	1 Brown Drive	506-153-6929

Additional_Service			
service_ID	**price**	**description**	**house_ID**
1	$85.47	Vivamus in felis eu sapien cursus vestibulum.	3
2	$492.69	Nam dui.	9
3	$311.46	In quis justo.	13
4	$158.14	Nullam sit amet turpis elementum ligula vehicula consequat.	4
5	$445.78	Praesent blandit.	17
6	$291.61	Aenean lectus.	5
7	$362.70	Donec semper sapien a libero.	10
8	$197.59	Nunc purus.	4
9	$194.87	Morbi ut odio.	5
10	$333.00	Maecenas rhoncus aliquam lacus.	17

Contract				
contract_ID	**monthly_fee**	**sDate**	**eDate**	**description**
1	$158.42	9/11/2019	7/4/2019	Proin at turpis a pede posuere nonummy.
2	$189.97	8/15/2019	12/1/2018	Quisque ut erat.
3	$287.62	11/7/2018	9/5/2019	Proin at turpis a pede posuere nonummy.
4	$323.30	9/20/2019	11/2/2018	Integer ac leo.
5	$337.06	5/30/2019	2/9/2019	Proin at turpis a pede posuere nonummy.
6	$179.88	6/14/2019	12/14/2018	Aenean auctor gravida sem.
7	$458.85	10/22/2019	7/24/2019	Quisque porta volutpat erat.
8	$367.14	6/19/2019	4/7/2019	Integer pede justo, lacinia eget, tincidunt eget, tempus vel, pede.
9	$172.96	7/21/2019	12/6/2018	Nulla nisl.
10	$338.85	1/23/2019	7/15/2019	Vestibulum quam sapien, varius ut, blandit non, interdum in, ante.

Income					
income_ID	**type**	**amount**	**date**	**description**	**contract_ID**
1	cash	$162.90	9/30/2019	Praesent blandit.	109
2	cash	$499.26	1/12/2019	In est risus, auctor sed, tristique in, tempus sit amet, sem.	254
3	bank deposit	$140.20	5/18/2019	Nulla tellus.	207
4	check	$77.76	11/20/2018	Maecenas ut massa quis augue luctus tincidunt.	3
5	bank deposit	$234.41	8/8/2019	Vivamus tortor.	182
6	bank deposit	$213.75	1/3/2019	Donec odio justo, sollicitudin ut, suscipit a, feugiat et, eros.	269
7	cash	$248.23	7/9/2019	In sagittis dui vel nisl.	152
8	money order	$486.72	4/13/2019	Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Duis faucibus accumsan odio.	89
9	bank deposit	$370.85	12/14/2018	Donec ut mauris eget massa tempor convallis.	129
10	check	$1.72	2/16/2019	Curabitur gravida nisi at nibh.	218

2.4. Sample Queries for Database

In Section 2.4, we will discuss how to design queries for the sample data given in the previous section. The relational database allows us three methods for writing queries which include: relational algebra, tuple calculus, and domain calculus. We will present the queries by retrieving data from our relations, or tables, creating in the relational model. Specific data retrievals or specific data retrievals will be described and then solved using all three methods.

2.4.1 Design Of Queries

In order to design the queries for our database it is important to know all the attributes and relations within the relational model. After defining all constraints, attributes, and keys we can solve any appropriate query using query language. The following section will present ten queries and three methods to solving the query.

2.4.2 Relational Algebra Expressions

Relational algebra is the first method in which we will design our queries. The expressions involved in relational algebra are a set of operations for retrieving tuples from a relational state. The expressions then describe the process for retrieving the tuples which means they are procedural. The order and nesting of the expressions is important for the query.

1. List all clients who have been under contract for past three months.

$$\Pi_{C.*}\left(\sigma_{\substack{c.clientID = h.clientID \wedge h.contractID = ct.contractID \wedge \\ ct.sDate <= f(\text{'Today,3 months'}) \wedge ct.eDate >= \text{'Today'}}}\left(Client^{c} \times House^{h} \times Contract^{ct}\right)\right)$$

2. List names of clients who have at least two houses under contract.

$$\Pi_{C.*}\left(\sigma_{\substack{c.clientID = h1.clientID \wedge h1.clientID = h2.clientID \wedge \\ h1.houseID \neq h2.houseID \wedge h1.contractID = ct.contractID \wedge \\ h1.contractID = h2.contractID \wedge ct.eDate > \text{'Today'}}}\left(Client^{c} \times House^{h1} \times House^{h2} \times Contract^{ct}\right)\right)$$

3. List all employees who have worked over 50 hours and were paid cash in the month of September (paid monthly at end of month).

$$\Pi_{e.*}\ (\sigma_{p.paymentID = c.paymentID \wedge p.hrs\_worked >= 50 \wedge p.Type = 'cash' \wedge p.date = '9\text{-}30\text{-}18'}\ Employee^{e} \times Payment^{p})$$

4. List houses that have a yearly contract and included more than one additional service.

$$\Pi_{h.*}\ (\sigma_{h.contractID = c.contractID \wedge a1.houseID = h.houseID \wedge a2.houseID = h.houseID \wedge a.serviceID \neq a2.serviceID}\ House^{h} \times Contract^{c} \times Additional\_Service^{a1} \times Additional\_Service^{a2})$$

5. List the houses worked on the date 3/6/18 and all employees who worked those houses.

$$\Pi_{h.ID,\ e.ID}\ (\sigma_{h.houseID = r.houseID \wedge c.routeID = r.routeID \wedge c.carID = a.carID \wedge a.employeeID = e.employeeID \wedge r.date = '3\text{-}6\text{-}18'}\ House^{h} \times Route^{r} \times Car^{c} \times Assign\_To^{a} \times Employee^{e})$$

6. List all cars that hold only tools made from 'Honda'.

$$\Pi_{h.*}\ (Car^{c} - \Pi_{c.*}\ (\sigma_{c.carID = t.carID \wedge t.brand \neq 'Honda'}\ Car^{c} \times Tool^{t}))$$

7. List all clients whose contract has a monthly fee of $100 or more.

$$\Pi_{c.*}\ (\sigma_{h.contractID = ct.contractID \wedge h.clientID = c.clientID \wedge ct.monthly\_fee > 100}\ Contract^{ct} \times Client^{c} \times House^{h})$$

8. List all employees who worked in specific car on 3/1/18 on route #4.

$$\Pi_{c.*}\ (\sigma_{c.routeID = r.routeID \wedge a.carID = c.carID \wedge a.employeeID = c.employeeID \wedge a.date = 3\text{-}1\text{-}18 \wedge r.routeID = 4}\ Route \times Car \times Assign\_to \times Employee)$$

9. List the second most expensive contract.

$$Result \leftarrow \Pi_{c1.*}\ (\sigma_{c1.monthlyfee > c2.monthlyfee}\ Contract_{c1} \times Contract_{c2})$$

$$\Pi_{c.*}\ (Result - \Pi_{c1.*}\ (\sigma_{c1.monthlyfee > c2.monthlyfee}\ Result_{c1} \times Result_{c2}))$$

10. List all the employees who worked with John Doe on dates between 3/1/18 and 3/5/18.

$$\Pi_{e.*}\ ((\sigma_{c.carID = a.carID \wedge a.employeeID = c.employeeID}\ Car \times Assign\_To \times Employee) \times \Pi_{c.carID}\ (\sigma_{c.carId = a.carID \wedge a.eID \; a.employeeID = e.employeeID \wedge E.name = \text{'John Doe'}}\ Car \times Assign\_to \times Employee))$$

2.4.3 Tuple Relational Calculus Expressions

Tuple relational calculus uses procedural expressions. It uses free variables (the variables that describe what the query will retrieve), bound variables (variables bounded by existential or universal quantifiers), and logical expressions with truth or false value.

1. List all clients who have been under contract for past three months.

{ c.* | client(c) ^ (∃h) (∃c) (house(h) ^ Contract(ct) ^
c.clientID = h.clientID ^
h.contractID = ct.contractID ^
ct.sDate <= pastMonth(now,3) ^
cs.eDate >= now) }

2. List names of clients who have at least two houses under contract.

{c.* | client(c) ^ ∃(h1) ∃(h2) ∃(ct) (house(h1) ^ house(h2) ^ contract(ct) ^
c.clientID = h1.clientID ^
c.clientID = h2.clientID ^
h1.contractID = ct.contractID ^
h2.contractID = ct.contractID ^
h1.houseID != h2.houseID ^
ct.eDate > 'now') }

3. List all employees who have worked over 50 hours and were paid cash in the month of september (paid monthly).

{e.* |employee(e) ^ ∃(p) (payment(p) ^
p.payment = e.payment ^
P.hrs-worked >= 50 ^
p.type = "cash" ^
p.date = '9-30-18') }

4. List houses that have a yearly contract and included more than one additional service.

{h.* | house(h) ^ ∃(c) ∃(a) ∃(a2) (contract (c) ^ add_service(a) ^
add_service(a2) ^
h.contractID = c.contractID ^
a.houseID = h.houseID ^
a2.houseID = h.houseID ^
a.serviceID != a2.serviceID) }

5. List the houses worked on the date 3/6/18 and all employees who worked those houses.

{<h.houseID, e.employeeID> | house(h) ^ ∃(r) ∃(c) ∃(a) ∃(e) (
route(r) ^ car(c) ^ assign_to(a) ^
employee(e) ^
h.houseID = r.houseID ^
c.routeID = r.routeID ^
c.carID = a.carID ^
A.employeeID = e.employeeID ^
R.date = '3-6-18') }

6. List all trucks that hold only tools made from specific "brand"

{c.* | car(c) ^ ∀(t) (tools(t) → ∃(c2) (car(c2) ^
c2.carID = t.carID ^
T.brand = "Honda") }

7. List all clients whose contract has a monthly fee of $100 or more.

{c.* | contract(c) ^ ∃(h) ∃(c1) (house(h) ^ client(c1) ^
c.contractID = h.contractID ^
h.clientID = c1.clientID ^
C.monthly_fee > 100) }

8. List all employees who worked in specific car on 3/1/18 on route #4.

{e.* | route(r) ^ ∃(c) ∃(a) ∃(e) (car(c) ^ assign_to ^ employee(e) ^
c.routeID = r.routeID ^
a.carID = c.carID ^
a.employeeID = e.employeeID ^
A.date = '3-01-18' ^
r.routeID = 4) }

9. List the second most expensive contract

{c.* | contract(c) ^ ∃(c2) **(** contract(c2) ^
c2.monthly_fee > c.monthly_fee ^
¬∃(c3)(contract(c3) ^
c3.monthly_fee > c2.monthly_fee)**)** }

10. List all the employees who worked with John Doe on dates between 3/1/18 and 3/5/18.

{e1.* | car(c1) ^ ∃(a1) ∃(e1) **(**assign_to(a1) ^ employee(e1) ^
c1.carID = a1.carID ^
a1.employeeID = e1.employeeID ^
∃(c) ∃(as) ∃(e) (car(c) ^ assign_to(as) ^ employee(e) ^
c.carID = as.carID ^
e.employeeID = as.employeeID ^
e.name = 'John Doe'**)** }

2.4.4 Domain Relational Calculus Expressions

Domain relational calculus is another branch of relational calculus that focuses on domains of the relation state. Each variable represents a single value with a tupe, instead of the entire tuple. Just like tuple relational calculus, it makes use of the existential (∃) and universal (∀) quantifiers and uses the same operators.

1. List all clients who have been under contract for past three months.

{<n, a, p, s> | client (n,a,p,s) ^ (∃ c) (Contract(_ , _ , f(today - 3 months), e
> 'today')}

2. List names of clients who have at least two houses under contract.

{<n> | client (c, n , _ , _, _) ^ (∃ ct) (Contract (ct , _ , _ , _ , > 'Today') ^
(∃ h1 ∃ h2) (House (h1, _ , _ , _ , _ , _ , _ , ct ,
c)
^ House (h2, _ , _ , _ , _ , _ , _ , ct, c)^ h1 != h2)}

3. List all employees who have worked over 50 hours and were paid cash in the month of september (paid monthly).

{<e> | employee (e, _ , _ , _ , _) ^ (∃ p) (Payment (p, 9/30/18, cash, _, 50,
_ ,e)}

4. List houses that have a yearly contract and included more than one additional service.

{<h> | house (h, _ , _ , _ , _ , _ , _ , ct , _) ^ (∃ ct) (Contract (ct , _ , _ , _) ^
(∃ a1 ∃ a2) (Additional_Service(a1 , _ , _ , h) ^
Additional_Service(a2 , _ , _ , h) ^
a1 != a2)}

5. List the houses worked on the date 3/6/18 and all employees who worked those houses.

{<h,n> | house (h, _ , _ , _ , _ , _ , _ , _ , _) ^ (∃ r) (Route (r, _ , _ , _ , h) ^
(∃ c) (Car (c , _ , _ , _ , r) ^
(∃ a) (Assign (c , 3/6/18, e) ^
(∃ e) (Employee (e , n , _ , _ , _)}

6. List all trucks that hold only tools made from 'Honda'.

{<c> | Car (c , _ , _ , _ , _) ^ (∃ t)(∀ b) (Tool (t, b , _ , _ , c) -> b = Honda}

7. List all clients whose contract has a monthly fee of $100 or more.

{<n> | Client (c, n, _ , _ , _) ^ (∃ h) (House (h, _ , _ , _ , _ , _ , _ , _ , _) ^
(∃ ct)(Contract(ct, >= 100, _ , _ , _)

8. List all employees who worked in specific car on 3/1/18 on route #4.

{<e,n> | Employee (e , n , _ , _ , _)^ (∃ c ∃ r ∃ a ∃ e) (Car (c , _ , _ , _ 4) ^
(Route (4, _ , _ , _ , _) ^
(Assign_To(a, c , 3/1/18, 3/1/18, e) ^
(Employee(e , n , _ , _ ,_))}

9. List the second most expensive contract

{ <c2> | Contract (c2, m2, _ , _ , _) ^ (∃ m1 ∃ c1) (Contract(c1, m1, _ , _ , _
)
^ (∃ m2 ∃ c2) (Contract(c3, m3, _ , _ , _
)
^ m2 > m1 ^ m2 < m3)}

10. List all the employees who worked with John Doe on dates between 3/1/18 and 3/5/18.

{<e,n> | Employee (e, n, _ , _ , _) ^ (∃ c) (Car(c, _ , _ , _ , _) ^
(∃ a) (Assign_To(a, c, 3/1/18, 3/5/18, e)
^
(∃ e2)(Employee(e2, John Doe, _ , _) ^
(∃ a2)(Assign_To(a2, c, 3/1/18, 3/5/18,
e2)}