Database Systems Project

CS 342

Derrick McKee

Dr. H. Wang, Professor

Computer Science Department

California State University, Bakersfield

November 29, 2010

Table of Contents

Phase IV

Phase V

Phase I

1.1) **Finding Facts and Techniques Used** - A fact finding technique that I used to gather data and operational data was personal experience with the company. As a consumer, I have used the Amtrak to transport myself to northern California. Another source I used was the company Amtrak's website to observe train scheduling, locations of train stations, and jobs onboard the train.

1.2) **Introduction to Organization** - As the nation's intercity passenger rail operator, Amtrak connects America in safer, greener and healthier ways. With 21,000 route miles in 46 states, the District of Columbia and three Canadian provinces, Amtrak operates more than 300 trains each day — at speeds up to 150 mph — to more than 500 destinations.

1.3) **Structure of the Organization** - The database for this project will derive from Amtrak's Pacific Surfliner branch of California and focus on the characteristics of the branches scheduling of trains, arrival and departure times of trains, and the persons on the trains. The database will also include information about the relationships between entities. For example, cars and train in which train 'pulls' cars, and train, employee, schedule in which each are 'assigned' to each other at a certain time.

1.4) **Itemized Description of Major Objects** - The objects train and cars are related to each other by 'pulls'. Schedule, Employee, and train are related to 'assigned to' in which employee and train are given a specific schedule. Finally, train and train station are related by 'enroute to' in which a train will depart train station A and arrive to train station B.

1.5) **Data Views and Operations** -The data views and operations for the user are solely based off train, the train's travel, employees, and how they are all related to each other by being assigned to a schedule.

2.1) **Entities**

2.1.1)

Name: Train
Description: The train engine
Attributes:

Name: TrainNumCode
Description: A number given to all the train engines to easily identify
Domain/Type: Number(5)
Value-Range: integers 1-99999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: TrainStartDate
Description: The starting date a train was created (Mo/Yr)
Domain/Type: Date

Value-Range: 0001-01-01 through 9999-12-31
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Multiple
Simple or Composite: Composite

Name: TrainEndDate
Description: The ending date of a train (Mo/Yr)
Domain/Type: Date
Value-Range: 0001-01-01 through 9999-12-31
Default Value: none
Null Value Allowed: Yes
Unique: No
Single or Multiple Value: Multiple
Simple or Composite: Composite

Candidate Keys: TrainNumCode
Primary Key: TrainNumCode
Strong/Weak Entity: Strong
Fields to be Indexed: TrainNumCode, TrainStartDate, TrainEndDate

2.1.2)

Name: Cars
Description: The identical cars that the Train Engine pulls
Attributes:

Name: CarNumCode
Description: A number given to all the train engines to easily identify
Domain/Type: Number(4)
Value-Range: integers 1-9999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: NumSeats
Description: The maximum number of seats in the car (2 sets of 2x10 seats separated by the aisle)
Domain/Type: Number(4)
Value-Range: integers 1-9999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: CarStartDate
Description: The starting date a car assigned to a train (Mo/Yr)
Domain/Type: Date

Value-Range: 0001-01-01 through 9999-12-31
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Multiple
Simple or Composite: Composite

Name: CarEndDate
Description: The ending date a car was assigned to a train (Mo/Yr)
Domain/Type: Date
Value-Range: 0001-01-01 through 9999-12-31
Default Value: none
Null Value Allowed: Yes
Unique: No
Single or Multiple Value: Multiple
Simple or Composite: Composite

Candidate Keys: CarNumCode
Primary Key: CarNumCode
Strong/Weak Entity: Strong
Fields to be Indexed: CarNumCode, NumSeats, CarStartDate, CarEndDate

2.1.3)

Name: Schedule
Description: The scheduling of a specific trip
Attributes:

Name: SchdleCode
Description: A number given to all the schedules to easily identify the trip
Domain/Type: Number(5)
Value-Range: 1-99999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: Schedule_Dept
Description: The departing date and time
Domain/Type: DateTime
Value-Range: 0001-01-01 00:00:00 through 9999-12-31 23:59:59
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Multiple
Simple or Composite: Composite

Name: Schedule_Arr
Description: The arrival date and time
Domain/Type: DateTime

Value-Range: 0001-01-01 00:00:00 through 9999-12-31 23:59:59
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Multiple
Simple or Composite: Composite

Candidate Keys: SchdleCode
Primary Key: SchdleCode
Strong/Weak Entity: Strong
Fields to be Indexed: SchdleCode

2.1.4)

Name: Employee
Description: Persons who work directly on the Train and Cars
Attributes:

Name: EmpCode
Description: A number given to all the employees to easily identify
Domain/Type: Number(5)
Value-Range: 1-99999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: Empfname
Description: The first name of the employee
Domain/Type: Varchar2(15)
Value-Range: 15 chars
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: Emplname
Description: The last name of the employee
Domain/Type: Varchar2(15)
Value-Range: 15 chars
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: EmpPhoneNum
Description: Employee's telephone number
Domain/Type: Varchar2(12)
Value-Range: 12 chars
Default Value: 0
Null Value Allowed: No
Unique: No
Single or Multiple Value: multiple
Simple or Composite: Simple

Candidate Keys: EmpCode
Primary Key: EmpCode
Strong/Weak Entity: Strong
Fields to be Indexed: EmpCode

2.1.5)

Name: SecurityGuard
Description: Person who works directly on the Train or Car
Attributes:

Name: SGcode
Description: A number given to all the employee of this department to easily identify
Domain/Type: Number(5)
Value-Range: integers 1-99999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: SGAddress
Description: The employee's address
Domain/Type: Varchar2(45)
Value-Range: 45 chars
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Single
Simple or Composite: Multiple

// This attribute was added later to implement topNavg
Name: SGSalary
Description: The employee's annual salary
Domain/Type: Number(5)
Value-Range: 1-99999
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Single

Simple or Composite: Multiple

Candidate Keys: SGCode
Primary Key: SGCode
Strong/Weak Entity: Strong
Fields to be Indexed: SGCode, SGAddress, SGSalary

2.1.6)

Name: Conductor
Description: Person who works directly on the Train or Car
Attributes:
Name: Condcode
Description: A number given to all the employee of this department to easily identify
Domain/Type: Number(5)
Value-Range: 1-99999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: CondAddress
Description: The employee's address
Domain/Type: Varchar2(45)
Value-Range: 45 chars
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Single
Simple or Composite: Multiple

Candidate Keys: CondCode
Primary Key: CondCode
Strong/Weak Entity: Strong
Fields to be Indexed: CondCode, CondAddress

2.1.7)

Name: Engineer
Description: Person who works directly on the Train or Car
Attributes:
Name: Engcode
Description: A number given to all the employee of this department to easily identify
Domain/Type: Number(5)
Value-Range: integers 1-99999

Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: EngAddress
Description: The employee's address
Domain/Type: Varchar2(45)
Value-Range: 45 chars
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Single
Simple or Composite: Multiple

Candidate Keys: EngCode
Primary Key: EngCode
Strong/Weak Entity: Strong
Fields to be Indexed: EngCode, EngAddress

2.1.8)

Name: TrainStation
Description: The train station at which trains and persons arrive and depart from
Attributes:

Name: StationName
Description: The train station's name
Domain/Type: Varchar2(15)
Value-Range: 15 chars
Default Value: Null
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Name: StationAddress
Description: The train station's address
Domain/Type: Varchar(25)
Value-Range: 25 chars
Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: Single
Simple or Composite: Multiple

Name: StationPhoneNum
Description: Employee's telephone number
Domain/Type: Varchar(11)
Value-Range: 11 chars

Default Value: none
Null Value Allowed: No
Unique: No
Single or Multiple Value: multiple
Simple or Composite: Simple

Name: StationCode
Description: A number given to all the train stations to easily identify
Domain/Type: Number(5)
Value-Range: 1-99999
Default Value: none
Null Value Allowed: No
Unique: Yes
Single or Multiple Value: Single
Simple or Composite: Simple

Candidate Keys: StationCode
Primary Key: StationCode
Strong/Weak Entity: Strong
Fields to be Indexed: StationName, StationAddress, StationPhoneNum, StationCode

2.2) **Relationship Set Description**

2.2.1)

Name: Pulls
Description: A train "pulls" the cars. The cars assigned to train
Entity Sets Involved: Train, Cars
Mapping Cardinality: 1:M
Participation Constraint: total/mandatory

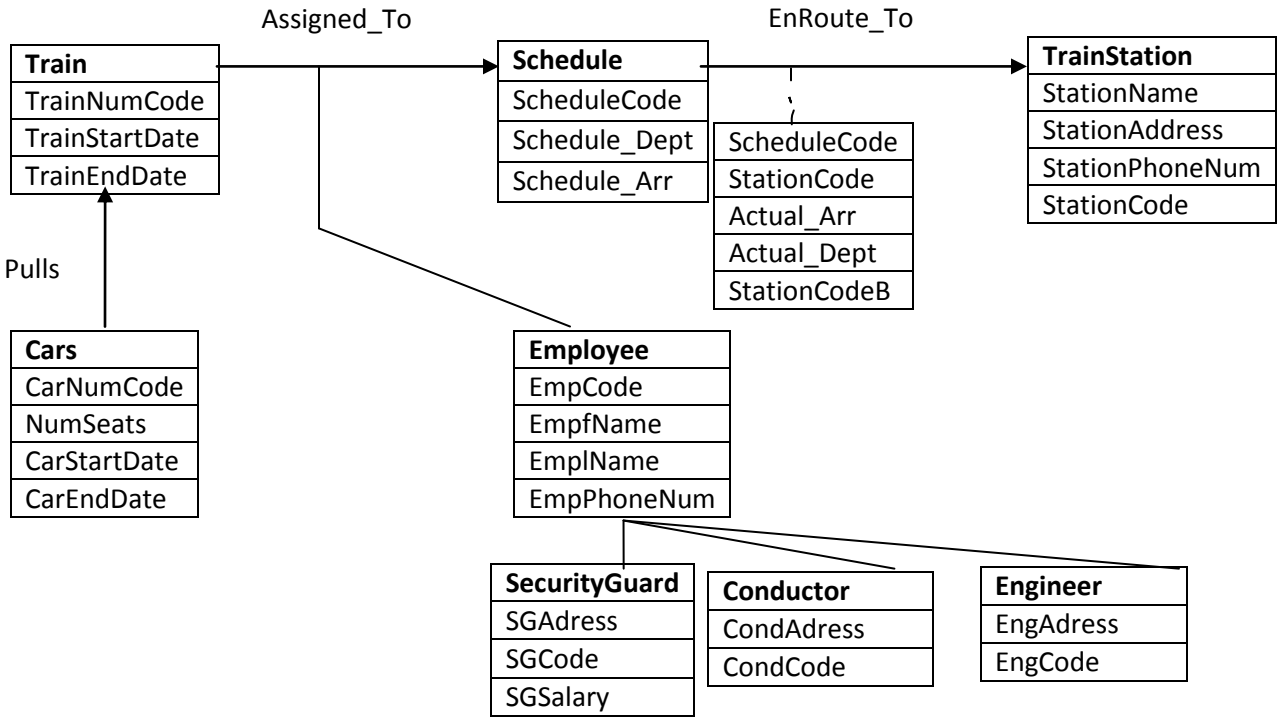
2.2.2)

Name: Assigned_To
Description: The entities Employee and Train are all assigned to Schedule
Entity Sets Involved: Train, Schedule
Mapping Cardinality: M:1
Participation Constraint: total/mandatory

2.2.3)

Name: EnRoute_To
Description: The attributes used are ScheduleCode, StationCode, Actual_Arr, Actual_Dept, and StationCodeB
Entity Sets Involved: Schedule, TrainStation
Mapping Cardinality: M:1
Participation Constraint: total/madatory

2.3) E-R Diagram



Phase II

3.1) **E-R model and relational model**

3.1.1) **Description** – The entity-relationship model is easy visual organization for a database design. Though, the database must be converted into a relational model before the construction of a database begins. The relational model was first exemplified by Ted Codd, an IBM researcher. He described the relational model as a set of relations with constraints set on a given domain or domains. The definition of a relational database schema is a set of relation schemas, each with a unique name. The purpose of this is to allow for a high degree of data to be manipulated which offers faster and easier conversion into the actual database.

3.1.2) **Comparison** – The entity-relational model is a visual description of an intended database. Features of the entity-relational model are entities, attributes, relations, and cardinality between entities. The relational model offers descriptive relationships between entities as a table with columns and rows. The domain constraint of each column is the name of the attribute. Each row of the table is a record in each table, also known as tuples.

3.1.3) **Conversion** – The conversion from the entity relational model to a relational model uses relations and column attributes. A relation may created for each strong entity type, using a unique attribute at a primary key for the relation and a combination of attributes. For weak entities, attributes of the weak entity are sent to the relation.

For all 1:1 binary relationship types can be created by mapping the primary key of the first entity and the primary key of the other, in which both attributes are candidate keys. Also, map the primary key of the first entity and use the second entity's primary key as an attribute.

For a 1:Many, map the primary key of the first entity into the second the other entities. Or, make a relation with the primary keys from each entity as attributes.

For a Many:Many relationship, a separate relationship must be created with the primary keys from each entity.

3.1.4) **Constraints** – An entity constraint is where no primary key can be null. Only an unknown value or unknown value at the time called may be null. Each primary key is unique which ensure the elements of each tuple to be unique. This is useful for comparisons and representations between relations. The definition of a referential constraint states that a foreign key must exist in a relation. The foreign key must match with a candidate key or must be altogether null. In addition to integral constraints, general constrains are additional rules specified by the user or database administrator which may constrain or restrict parts of the database.

3.2) **Relational Database**

3.2.1) **Train Relation Attributes**

TrainNumCode

Domain/Type: A positive integer: 1 - 99999 (cannot be null)

TrainStartDate

Domain/Type: valid Date (cannot be null)

TrainEndDate

Domain/Type: valid Date (can only be null if Train is still in use)

Constraints:

Primary Key: TrainNumCode – must be unique and not null.

Rule: TrainStartDate and TrainEndDate must be valid Date, but TrainEndDate may be null if train is still in use.

Candidate Key:

TrainNumCode

3.2.2) **Cars Relation Attributes**

CarNumCode

Domain/Type: A positive integer: 1 - 9999 (cannot be null)

NumSeats

Domain/Type: A positive integer: 1 – 9999, cannot be null

CarStartDate

Domain/Type: valid Date (cannot be null)

CarEndDate

Domain/Type: valid Date (can only be null if Car is still in use)

Constraints:

Primary Key: CarNumCode – must be unique a not null.

Rule: CarStartDate and CarEndDate must be valid Date, but CarEndDate may be null if train is still in use.

Candidate Key:

CarNumCode

3.2.3) **Schedule Relation Attributes**

ScheduleCode

Domain/Type: A positive integer: 1-99999 (cannot be null)

Schedule_Arr

Domain/Type: Varchar2(21) (cannot be null)

Schedule_Dept

Domain/Type: Varchar2(21) (cannot be null)

Constraints:

Primary Key: SchdleCode – must be unique and not null

Rule: All dates must be valid Date

Candidate Key:

SchdleCode

3.2.4) **Employee Relation Attributes**

EmpCode

Domain/Type: Number(5) : 1 – 99999, cannot be null.

EmpfName

Domain/Type: A string that holds up to 15 chars, cannot be null.

EmplName

Domain/Type: A string that holds up to 15 chars, cannot be null.

EmpPhoneNum

Domain/Type: A string that holds up to 12 chars, cannot be null.

Constraints:

Primary Key: EmpCode must have unique value

Foreign Key: EmpCode, EmpfName, EmplName, EmpPhoneNum are foreign keys from their respective relations

Candidate Key:

EmpCode

3.2.5) **SecurityGuard Relation Attributes**

SGCode

Domain/Type: Number(5) : 1 – 99999, cannot be null.

SGAddress

Domain/Type: A string containing 45 characters, cannot be null

SGSalary

Domain/Type: Number(5) : 1 – 99999, cannot be null.

Constraints:

Primary Key: SGCode must have unique value

Foreign Key: SGCode is a foreign key from their respective relations

Candidate Key:

SGCode

3.2.6) **Conductor Relation Attributes**

CondCode

Domain/Type: Number(5) : 1 – 99999, cannot be null.

CondAddress

Domain/Type: A string that holds 45 characters and cannot be null.

Constraints:

Primary Key: CondCode must have unique value

Foreign Key: CondCode is a foreign key from their respective relations

Candidate Key:

CondCode

3.2.7) **Engineer Relation Attributes**

EngCode

Domain/Type: Number(5) : 1 – 99999, cannot be null.

EngAddress

Domain/Type: A string that holds 45 characters, cannot be null.

Constraints:

Primary Key: EndCode must have unique value

Foreign Key: EngCode is a foreign key from their respective relations

Candidate Key:

EngCode

3.2.9) **TrainStation Relation Attributes**

StationName

Domain/Type: A string that holds up to 15 chars, cannot be null

StationPhoneNum

Domain/Type: A string that holds up to 11 chars, cannot be null

StationAddress

Domain/Type: A string that holds up to 25 chars, cannot be null

StationCode

Domain/Type: Number(5) : 1 - 99999, cannot be null

Constraints:

Primary Key: StationCode must have unique value

Candidate Key:

StationCode

3.2.10) **Pulls Relation Attribute**

TrainNumCode

Domain/Type: Number(5) : 1 - 99999 (cannot be null)

CarNumCode

Domain/Type: Number(4) : 1 - 9999 (cannot be null)

StartDate

Domain/Type: valid Date (cannot be null)

EndDate

Domain/Type: valid Date

Constraints:

Rule: The EndDate cannot be an entry less than the StartDate, and can also be null if the train is presently pulling the car

Foreign Key: TrainNumCode, CarNumCode and must both exist in their respective relations

Candidate Key:

None

3.2.11) **Assigned_To Relation Attribute**

EmpCode

Domain/Type: Number(5) : 1 – 99999 (cannot be null)

TrainNumCode

Domain/Type: Number(5) : 1 – 99999 (cannot be null)

ScheduleCode

Domain/Type: Number(5) : 1 – 99999 (cannot be null)

Constraints:

Foreign Key: EmpCode, TrainNumCode, ScheduleCode and must all exist in their respective relations.

Candidate Key:

None

3.2.12) **EnRoute_To Relation Attribute**

ScheduleCode

Domain/Type: Number(5) : 1 – 99999 (cannot be null)

StationCode

Domain/Type: Number(5) : 1 – 99999, cannot be null

Actual_Dept

Domain/Type: valid Date

Actual_Arr

Domain/Type: valid Date

StationCodeB

Domain/Type: valid Date, cannot be null or equal to StationCode

Constraints:

Foreign Key: ScheduleCode, StationCode, Actual_Arr, Actual_Dept , ScheduleCodeB must all exist in their respective relations.

Rule: All dates must be valid Date

Candidate Key:

None

3.3) **Relation Instances**

3.3.1) Train(TrainNumCode, TrainStartDate, TrainEndDate)

TrainNumCode	TrainStartDate	TrainEndDate
01	01/01/01	04/03/06
02	01/02/01	05/04/06
03	02/04/01	03/01/09
04	02/04/01	
05	03/19/03	
06	06/21/02	09/12/09
07	04/04/04	
08	12/12/01	
09	11/11/01	11/11/09
10	07/23/09	

3.3.2) Cars(CarNumCode, NumSeats, CarStartDate, CarEndDate)

CarNumCode	NumSeats	CarStartDate	CarEndDate
01	40	01/01/01	
02	40	01/02/01	
11	40	02/01/02	
12	40	01/02/01	01/02/09
13	40	01/02/01	03/13/09
24	40	04/01/01	04/23/01
25	40	03/06/01	05/11/08
26	40	07/23/01	
30	40	03/03/03	04/04/04
31	400	05/04/03	06/05/04

3.3.3) Schedule(ScheduleCode, Schedule_Dept, Schedule_Arr)

ScheduleCode	Schedule_Dept	Schedule_Arr
01	01/01/01 6:00	01/01/01 8:00
02	01/01/01 8:05	01/01/01 10:00
03	01/01/01 10:05	01/01/01 12:00
04	01/01/01 12:05	01/01/01 15:00
05	01/01/01 15:05	01/01/01 17:00
06	01/01/01 17:10	01/01/01 15:00
07	01/02/01 6:00	01/02/01 7:00
08	01/02/01 7:05	01/02/01 7:55
09	01/02/01 8:00	01/02/01 8:55
10	01/02/01 9:00	01/02/01 9:55
11	01/02/01 10:00	01/02/01 10:55

3.3.4) Employee(EmpCode, EmpfName, EmplName, EmpPhoneNum)

EmpCode	EmpfName	EmplName	EmpPhoneNum
001	Ben	Neb	6613432222
002	Paul	Lap	7886564434
011	Ray	Sun	3232225555
012	Sara	Miller	2223334444
101	Taylor	Miller	2223334444
102	Zach	Ice	8886667878
111	Annie	Gracie	9871234444
201	Megan	Harris	8583950000
202	Derrick	McKee	6618098888
233	John	Doe	9990009999

3.3.5) SecurityGuard(SGCode, SGAddress)

SGCode	SGAddress
001	343 B St. Bakersfield, CA 93311
002	4434 P St. Bakersfi8eld, CA 93311
011	5555 Radiation Ave Bakersfield, CA 93311
012	234 Couple St. Azusa, CA 92123
014	0101 Binary Ct. Bakersfield, CA 93311
022	222 S St. Bakersfield, CA 93312
055	5555 S St. Bakersfield, CA 93312

3.6) Conductor(CondCode, CondAddress)

CondCode	CondAddress
101	234 Couple St. Azusa, Ca 92123
102	7878 Cold St. Bakersfield, CA 93312
111	303 Dog St. Bakersfield, CA 93312
112	911 Space Ct. Bakersfield, CA 93311
123	333 Shave St. Bakersfield, CA 93313
150	150 Money Ave Bakersfield, CA 93312
199	999 Even St. Bakersfield, CA 93312

3.3.7) Engineer(EngCode, EngAddress)

EngCode	EngAddress
201	Girl Ct. Bakersfield, CA 93312
202	Boy Ct. Bakersfield, CA 9312
233	Template St. Bakersfield, CA 93311
234	Template St. Bakersfield, CA 93311
235	Template St. Bakersfield, CA 93311
288	Buzz St. Bakersfield, CA 93311
290	Washington St. Bakersfield, CA 93311

3.3.8) TrainStation(StationName, StationPhoneNum, StationAddress, StationCode)

StationName	StationPhoneNum	StationAddress	StationCode
San Diego	8585558888	858 5th St. San Diego, CA	8
Oceano	8581114444	1144 1st St. Ocean, CA	14
Santa Anna	6667775656	56 5th St. Santa Anna,CA	5
Solana Beach	8582229999	2nd St. Solana Beach, CA	2
Anaheim	3434443333	3rd St. Anaheim, CA	3
Ventura	7771117777	7th St. Ventura, CA	7
Los Angeles	6669996666	6th St. Los Angeles	6

3.3.9) Pulls(TrianNumCode, CarNumCode, StartDate, EndDate)

TrainNumCode	CarNumCode	StartDate	EndDate
01	01	01/01/01	01/01/02
01	22	01/01/01	
01	02	01/02/01	01/01/05
03	01	01/04/02	
03	03	03/13/03	
02	13	01/02/01	05/01/05
02	12	01/02/01	

3.3.10) Assigned_To(EmpCode, TrainNumCode, ScheduleCode)

EmpCode	TrainNumCode	ScheduleCode
01	01	01
01	01	02
02	01	03
101	01	01
101	01	02
101	01	03
101	01	04
101	01	05
201	01	01
201	01	02
201	01	03
201	01	04
201	01	05
02	02	01
02	02	02
102	02	01
102	02	02
202	02	02

3.3.11) EnRoute_To(ScheduleCode, StationCode, Actual_Dept, Actual_Arr, StationCodeB)

ScheduleCode	StationCode	Actual_Dept	Actual_Arr	StationCodeB
01	5	6:00AM	7:55AM	6
02	6	8:05AM	10:00AM	5
03	5	10:05AM	11:55AM	6
04	6	12:30PM	3:00PM	5
05	5	3:05PM	5:00PM	6
06	6	5:05PM	7:00PM	5
07	3	6:00AM	6:55AM	6
08	6	7:00AM	7:55AM	3
09	3	8:00AM	8:55AM	6
10	6	9:00AM	9:55AM	3

List of Relations

Train(TrainNumCode, TrainStartDate, TrainEndDate)

Cars(CarNumCode, NumSeats, CarStartDate, CarEndDate)

Schedule(ScheduleCode, Schedule_Dept, Schedule_Arr)

Employee(EmpCode, EmpfName, EmplName, EmpPhoneNum)

SecurityGuard(SGCod, SGAddress)

Conductor(CondCode, CondAddress)

Engineer(EngCode, EngAddress)

TrainStation(StationName, StationPhoneNum, StationAddress, StationCode)

Pulls(TrianNumCode, CarNumCode, StartDate, EndDate)

Assigned_To(EmpCode, TrainNumCode, ScheduleCode)

EnRoute_To(ScheduleCode, StationCode, Actual_Dept, Actual_Arr, StationCodeB)

4) **Queries**

1. List Engineers who live on Template St. Bakersfield, CA 93311
2. List employees who are assigned to TrainNum 1
3. List conductor en route to Anaheim
4. List Engineers who have worked on TrainNum 2
5. List all the Trains scheduled during 12:00PM-3:00PM
6. List all the cars pulled by TrainNum 1
7. List all working Trains
8. List all working Cars
9. List the cities that TrainNum 1 travel to
10. List the security guard that is assigned to TrainNum 2

5) **Query Representation**

1. List Engineers who live on Template St. Bakersfield, CA 93311

Relational algebra:

$\pi_{Engineer}(\sigma_{(EngAddress = 'Template\ St.\ Bakersfield,\ CA\ 93311')} Engineer)$

Tuple relational calculus:

{e | Engineer(e) ^ e.EngAddress = 'Template St. Bakersfield, CA 93311'}

Domain relational calculus:

{<ec, ef, el>| Engineer(ec, ef, el,_, 'Template St. Bakersfield, CA 93311)}

2. List Employees assigned to TrainNum 1

Relational algebra:

$\pi_{Employee}(\sigma_{(A.EmpCode = E.EmpCode)} Assign\_To\ A * Employee\ E)$

Tuple relational calculus:

{e | Employe(e) ^ (∃a)(Assign_To(a) ^ a.EmpCode=e.EmpCode ^ a.TrainNum = 01)}

Domain Relational Calculus:

{<ec, ef, el, ep>| Employee(ec, ef, el, ep) ^ (∃a)(Assign_To(ec, '01', _)}

3. List conductor en route to Anaheim

Relational algebra:

$\pi_{Conductor}$ (σ Assign_To a * Conductor c * TrainStation t * EnRoute_To e)

(c.EmpCode = e.EmpCode ^ a.ScheduleCode = e.ScheduleCode ^ e.StationCode = t.StationCode ^ t.StationName = 'Anaheim')

Tuple relational calculus:

{c | Conductor(c) ^ (∃a)(∃t)(∃e)(Assign_To(a)^TrainStation(t)^EnRoute_To(e) ^ c.CondCode=a.EmpCode ^ a.ScheduleCode=e.ScheduleCode ^ e.StationCode=t.StationCode ^ t.StationName= 'Anaheim')}

Domain relational calculus:

{<cc, cf, cl>| Conductor(cc, cf, cl,_,_) ^ (∃s)(Assign_To(cc,_,s) ^ EnRoute_To(s,_,_,_,'Anaheim'))}

4. List Engineers that have worked on TrainNum2

Relational algebra:

$\pi_{Engineer}(\sigma_{(e.EmpCode = a.EmpCode\ ^\ a.TrainNumCode = 2)} Engineer\ e * Assigned\_To\ a)$

Tuple relational calculus:

{e | Engineer(e) ^ (∃a)(Assign_To(a) ^ a.EmpCode=e.EngCode ^ a.TrainNumCode = '02')}

Domain relational calculus:

{<ec> | Engineer(ec,_,_,_) ^ Assign_To(ec, '02', _)}

5. List all the trains scheduled during 12:00-3:00PM

Relational algebra:

π_{Train} (σ Train t * Schedule s * Assigned_To a)

(s.ScheduleArr>'12:00' ^ s.ScheduleDept<'15:00' ^ s.ScheduleCode = a.ScheduleCode ^ t.TrainNumCode = a.TrainNumCode)

Tuple relational calculus:

{t | Train(t) ^ (∃a)(∃s)(Assign_To(a) ^ Schedule(s) ^ s.ScheduleArr > '12:00' ^ s.ScheduleDept <'15:00' ^ s.ScheduleCode = a.ScheduleCode)}

Domain relational calculus:

{<t> | Train(t,_,_) ^ (∃s)(Assign_To(_,t,s) ^ Schedule(s, >12:00, <15:00))}

6. List all cars pulled by TrainNum 1

Relational algebra:

$\pi_{Cars}(\sigma_{(p.TrainNumCode = 1)}$ Cars c)

Tuple relational calculus:

{c | Car(c) ^ (∃p)(Pulls(p) ^ p.TrainNumCode = 01)

Domain relation calculus:

{<c> | Car(c,_,_,_) ^ Pulls(01, c, _, _)}

7. List all working trains

Relational algebra:

π_{Train}(σ $_{(t.EndDate = null)}$ Train t)

Tuple relational calculus:

{t | Train(t) ^ t.EndDate = null}

Domain relation calculus:

{<t>| Train(t,_, null)}

8. List all working cars

Relational algebra:

π_{Cars}(σ $_{(t.EndDate = null)}$ Cars c)

Tuple relational algebra:

{c | Cars(c) ^ c.EndDate = null}

Domain relation calculus:

{<c>| Cars(c,_, null)}

9. List the cities that TrainNum 1 has ever traveled to

Relational algebra:

$\pi_{TrainStation}$(σ Trianstation st * Schedule s * EnRoute_To e)

(s.TrainNumCode = 1 ^ s.ScheduleCode = e.ScheduleCode ^ e.StationCode = st.StationCode)

Tuple relational algebra:

{st | TrainStation(st) ^ (∃s)(∃e) (Schedule(s)^ EnRoute_To(e) ^ s.TrainNumCode='01' ^ s.ScheduleCode=e.ScheduleCode ^ e.StationCode = st.StationCode) }

Domain relational algebra:

{<c> | TrainStation(_,_,_, c) ^ (∃s)(EnRoute_To(s,c,_,_,_)) }

10. List the Security Guards assigned to TrainNum 2

Relational algebra:

$\pi_{SecurityGuard}$(σ $_{(a.EmpCode = sg.SGCode \wedge a.TrainNumCode = 2)}$ SecurityGuard sg * Assigned_To a)

Tuple relational algebra:

{sg | SecurityGuard(sg) ^ (∃a)(Assigned_To(a) ^ a.EmpCode = sg.SGCode ^ a.TrainNumCode = '02') }

Domain relation algebra:

{<sgc>| SecurityGuard(sgc,_,_,_,_) ^ Assigned_To(sgc, '02', _) }

Phase III

1) Purpose of SQL and Functionality

The Structured Query Language, or SQL, is a universal foundation database language used in many Database Management Systems (DBMS). It allows the database programmer to understand the specifics of how data is physically stored and provides a basis for updating, creating, and extracting data. SQL also provides the capability to perform simple to complex queries. SQL uses two roles, the Data Definition Language (DDL) and the Data Manipulation Language (DML). DDL allows the programmer to define the structure of the database, where as the DML is used to manipulate (update, insert, delete) data inside the database.

2) Oracle DBMS Schema Objects

Schema - A schema is a collection of database objects owned by the database user. Each object is a logical structure that directly references data from the database. Some structures are tables, views, and indexes.

Tables- Tables are the common unit of data storage in an Oracle database. Tables hold data from all users in a row by column structure. Each column in a table is the different data types of information where rows contain all the instances.

Syntax

CREATE TABLE TableName
{(columnName dataType [**NOT NULL**] [**UNIQUE**]
[**DEFAULT** defaultOption] [**CHECK** (searchCondition)] [,…]}
[**PRIMARY KEY** (listOfColumns),]
{[**UNIQUE** (listOfColumns)] [,…]}
{[**FOREIGN KEY** (listOfForeignKeyColumns)
REFERENCES ParentTableName [(listOfCandidateKeyColumns)]
[**MATCH** {**PARTIAL** | **FULL**}
[**ON UPDATE** referentialAction]
[**ON DELETE** referentialAction]][,…]}
{[**CHECK** (searchCondition)] [,…]});

Indexes- Indexes are structures associated with tables, and optional. In Oracle, an index provides an access path to requested table data in the database efficiently. After an index is created it can be automatically maintained by Oracle in which any change to table data is recorded.

Syntax

CREATE [**UNIQUE**] **INDEX** IndexName
ON TableName (columnName [**ASC** | **DESC**] [,…]);

Views- A view in SQL terminology is a single table that is derived by other tables. A view doesn't necessarily exist in physical form, rather it is considered a virtual table. A view acts like a table, but rather information is derived from tables which can be referenced easily by the database user. The tables used by a view are called the defining tables.

Syntax

CREATE VIEW Viewname [(newColumnName[,…])]
AS subselect [**WITH** [**CASCADED** | **LOCAL**] **CHECK OPTION**]

3) Relational Schema Objects and Contents

- djm_Assigned_To Assigned_To relation
- djm_Cars Cars relation
- djm_Conductor Conductor relation
- djm_Employee Employee relation
- djm_EnRoute_To EnRoute_To relation
- djm_Engineer Engineer relation
- djm_Pulls Pulls relation
- djm_Schedule Schedule relation
- djm_SecurityGuard SecurityGuard relation
- djm_Train Train relation
- djm_TrainStation TrainStation relation

Schemas and Instances for each relation

djm Assigned To

```
CS342 SQL> desc djm_Assigned_To;

 Name                                                                    Null?    Type

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

 EMPCODE                                                                 NOT NULL NUMBER(9)

 TRAINNUMCODE                                                             NOT NULL NUMBER(9)

 SCHEDULECODE                                                             NOT NULL NUMBER(9)


CS342 SQL> select * from djm_Assigned_To;


   EMPCODE TRAINNUMCODE SCHEDULECODE

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

         1            1            1

         1            1            2

         2            1            3

       101            1            1

       101            1            2

       101            1            3

       101            1            4

       101            1            5

       201            1            1

       201            1            2

       201            1            3
```

```
       201            1            4
       201            1            5
         2            2            1
         2            2            2
       102            2            1
       102            2            2
       202            2            2

18 rows selected.
```

djm_Cars

```
CS342 SQL> desc djm_Cars;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- -------------------------------------
--------
 CARNUMCODE                                                              NOT NULL NUMBER(4)
 NUMSEATS                                                                NOT NULL NUMBER(4)
 CARSTARTDATE                                                            NOT NULL DATE
 CARENDDATE                                                                       DATE


CS342 SQL> select * from djm_Cars;

CARNUMCODE   NUMSEATS CARSTARTD CARENDDAT
---------- ---------- --------- ---------
         1         40 01-JAN-01
         2         40 02-JAN-01
        11         40 02-JAN-02
        12         40 02-JAN-01 02-JAN-09
        13         40 02-JAN-01 13-MAR-09
        24         40 04-JAN-01 23-APR-01
        25         40 06-MAR-01 05-NOV-08
        26         40 23-JUL-01
        30         40 03-MAR-03 04-APR-04
        31         40 04-MAY-03 04-JUN-04

10 rows selected.
```

djm Conductor

```
CS342 SQL> desc djm_Conductor;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------
--------

 CONDCODE                                                                NOT NULL NUMBER(5)

 CONDADDRESS                                                             NOT NULL VARCHAR2(45)


CS342 SQL> select * from djm_Conductor;


  CONDCODE CONDADDRESS
---------- --------------------------------------------
       101 234 Couple St. Azusa, CA 92123

       102 7878 Cold St. Bakersfield, CA 93312

       111 303 Dog St. Bakersfield, CA 93312
```

djm Employee

```
CS342 SQL> desc djm_Employee;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------
--------

 EMPCODE                                                                 NOT NULL NUMBER(5)

 EMPFNAME                                                                NOT NULL VARCHAR2(15)

 EMPLNAME                                                                NOT NULL VARCHAR2(15)

 EMPPHONENUM                                                             NOT NULL VARCHAR2(12)


CS342 SQL> select * from djm_Employee;


   EMPCODE EMPFNAME        EMPLNAME        EMPPHONENUM
---------- --------------- --------------- ------------
         1 Ben             Neb             6613432222

         2 Paul            Lap             7886564434

        11 Ray             Sun             3232225555

        12 Sara            Miller          2223334444

       101 Taylor          Miller          2223334444
```

```
       102 Zach            Ice             8886667878
       111 Annie           Gracie          9871234444
       201 Megan           Harris          8583950000
       202 Derrick         McKee           6618098888
       233 John            Doe             9990009999

10 rows selected.
```

djm_EnRoute_To

```
CS342 SQL> desc djm_EnRoute_To;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------
--------
 SCHEDULECODE                                              NOT NULL NUMBER(5)
 STATIONCODE                                               NOT NULL NUMBER(5)
 ACTUAL_DEPT                                                        VARCHAR2(21)
 ACTUAL_ARR                                                         VARCHAR2(21)
 STATIONCODEB                                              NOT NULL NUMBER(5)

CS342 SQL> select * from djm_EnRoute_To;

SCHEDULECODE STATIONCODE ACTUAL_DEPT          ACTUAL_ARR            STATIONCODEB
------------ ----------- -------------------- -------------------- ------------
           1           5 6:00AM               7:55AM                          6
           2           6 8:05AM               10:00AM                         5
           3           5 10:05AM              11:55AM                         6
           4           6 12:30PM              3:00PM                          5
           5           5 3:05PM               5:00PM                          6
           6           6 5:05PM               7:00PM                          5
           7           3 6:00AM               6:55AM                          6
           8           6 7:00AM               7:55AM                          3
           9           5 8:00AM               8:55AM                          6
          10           5 9:00AM               9:55AM                          3

10 rows selected.
```

djm_Engineer

```
CS342 SQL> desc djm_Engineer
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------
--------

 ENGCODE                                                                 NOT NULL NUMBER(5)

 ENGADDRESS                                                              NOT NULL VARCHAR2(45)


CS342 SQL> select * from djm_Engineer;


   ENGCODE ENGADDRESS
---------- --------------------------------------------

       201 Girl Ct. Bakersfield, CA 93312

       202 Boy Ct. Bakersfield, CA 93312

       233 Template St. Bakersfield, CA 93311
```

djm_Pulls

```
CS342 SQL> desc djm_Pulls;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------
--------

 TRAINNUMCODE                                                            NOT NULL NUMBER(9)

 CARNUMCODE                                                              NOT NULL NUMBER(9)

 STARTDATE                                                               NOT NULL DATE

 ENDDATE                                                                          DATE


CS342 SQL> select *from djm_Pulls;


TRAINNUMCODE CARNUMCODE STARTDATE ENDDATE
------------ ---------- --------- ---------

           1          1 01-JAN-01 01-JAN-02

           1          2 02-JAN-01 01-JAN-05

           3          1 04-JAN-02

           3         25 13-MAR-03

           2         13 02-JAN-01 01-MAY-05

           2         12 02-JAN-01
```

```
6 rows selected.
```

djm_Schedule

```
CS342 SQL> desc djm_Schedule;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------
--------
 SCHEDULECODE                                                            NOT NULL NUMBER(5)
 SCHEDULE_DEPT                                                           NOT NULL VARCHAR2(21)
 SCHEDULE_ARR                                                            NOT NULL VARCHAR2(21)

CS342 SQL> select * from djm_Schedule
  2  ;

SCHEDULECODE SCHEDULE_DEPT         SCHEDULE_ARR
------------ -------------------- --------------------
           1 01/01/01 6:00AM      01/01/01 8:00AM
           2 01/01/01 8:05AM      01/01/01 10:00AM
           3 01/01/01 10:05AM     01/01/01 12:00PM
           4 01/01/01 12:05PM     01/01/01 3:00PM
           5 01/01/01 3:05PM      01/01/01 5:00PM
           6 01/01/01 5:10AM      01/01/01 7:00PM
           7 01/01/01 6:00AM      01/02/01 7:00AM
           8 01/01/01 7:05AM      01/02/01 7:55AM
           9 01/01/01 8:00AM      01/02/01 8:55AM
          10 01/01/01 9:00AM      01/02/01 9:55AM
          11 01/02/01 10:00AM     01/02/01 10:55AM

11 rows selected.
```

djm_SecurityGuard

```
CS342 SQL> desc djm_SecurityGuard
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------
--------
 SGCODE                                                                  NOT NULL NUMBER(5)
 SGADDRESS                                                               NOT NULL VARCHAR2(45)
```

```
CS342 SQL> select * from djm_SecurityGuard;

    SGCODE SGADDRESS
---------- --------------------------------------------
         1 343 B St. Bakersfield, CA 93311
         2 4434 P St. Bakersfield, CA 93311
        11 5555 Radiation Ave Bakersfield, CA 93311
        12 234 Couple St. Asuza, CA 92123
```

djm Train

```
CS342 SQL> desc djm_Train;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------
--------
 TRAINNUMCODE                                                            NOT NULL NUMBER(5)
 TRAINSTARTDATE                                                          NOT NULL DATE
 TRAINENDDATE                                                                     DATE

CS342 SQL> select * from djm_Train;

TRAINNUMCODE TRAINSTAR TRAINENDD
------------ --------- ---------
           1 01-JAN-01 03-APR-06
           2 02-JAN-01 04-MAY-06
           3 02-FEB-01 01-MAR-09
           4 04-FEB-01
           5 19-MAR-03
           6 21-JUN-02 21-SEP-09
           7 04-APR-04
           8 12-DEC-01
           9 11-NOV-01 11-NOV-09
          10 23-JUL-09

10 rows selected.
```

djm_TrainStation

```
CS342 SQL> desc djm_TrainStation;
 Name                                                                    Null?    Type
 ----------------------------------------------------------------------- -------- ------------------------------------
--------

 STATIONNAME                                                             NOT NULL VARCHAR2(15)
 STATIONPHONENUM                                                         NOT NULL VARCHAR2(11)
 STATIONADDRESS                                                          NOT NULL VARCHAR2(25)
 STATIONCODE                                                             NOT NULL NUMBER(5)


CS342 SQL> select * from djm_TrainStation;

STATIONNAME     STATIONPHON STATIONADDRESS            STATIONCODE
--------------- ----------- ------------------------- -----------
San Diego       8585558888  858 5th St. San Diego, CA           8
Oceano          8581114444  1144 1st St. Oceano, CA            14
Santa Anna      6667775656  56 5th St. Santa Anna, CA           5
Solana Beach    8582229999  2nd St. San Diego, CA               2
Anaheim         3434443333  3rd St. Anaheim, CA                 3
Ventura         7771117777  7th St. Ventura, CA                 7
Los Angeles     6669996666  6th St. Los Angeles, CA             6

7 rows selected.
```

4) SQL Queries

Query 1 – List Engineers who live on Template St. Bakersfield, CA 93311

```
CS342 SQL> @q1.sql

   ENGCODE ENGADDRESS
---------- --------------------------------------------
       233 Template St. Bakersfield, CA 93311


CS342 SQL> ;
  1  select unique e.* from djm_Engineer e
  2* where e.EngAddress = 'Template St. Bakersfield, CA 93311'
```

Query 2 – List Employees assigned to TrainNum 1

```
CS342 SQL> @q2.sql

   EMPCODE EMPFNAME        EMPLNAME        EMPPHONENUM
---------- --------------- --------------- ------------
         1 Ben             Neb             6613432222
       101 Taylor          Miller          2223334444
         2 Paul            Lap             7886564434
       201 Megan           Harris          8583950000


CS342 SQL> ;
  1  select unique e.*
  2  from djm_Employee e inner join djm_Assigned_To a on (e.EmpCode=a.EmpCode
  3*            and a.TrainNumCode = 1)
```

Query 3 – List conductor en route to Los Angeles

```
CS342 SQL> @q3.sql

   EMPCODE EMPFNAME        EMPLNAME        EMPPHONENUM
---------- --------------- --------------- ------------
       101 Taylor          Miller          2223334444
       102 Zach            Ice             8886667878


CS342 SQL> ;
  1  select unique e.*
  2  from djm_Employee e inner join djm_Conductor c on (e.EmpCode = c.CondCode)
  3     inner join djm_Assigned_To a on (e.EmpCode = a.EmpCode)
  4     inner join djm_Enroute_To en on (a.ScheduleCode = en.ScheduleCode
  5*      and en.StationCodeB = 6)
```

Query 4 – List Engineers who have worked on TrainNum 2

```
CS342 SQL> @q4.sql

   EMPCODE EMPFNAME        EMPLNAME        EMPPHONENUM
---------- --------------- --------------- ------------
       202 Derrick         McKee           6618098888


CS342 SQL> ;
  1  select unique e.*
  2  from djm_Employee e inner join djm_Engineer eng on (e.EmpCode = eng.EngCode)
  3     inner join djm_Assigned_To a on (eng.EngCode=a.EmpCode
  4*    and a.TrainNumCode = 02)
```

Query 5 – List all the Trains scheduled during 12:00PM-3:00PM

```
CS342 SQL> @q5.sql

TRAINNUMCODE SCHEDULECODE SCHEDULE_DEPT         SCHEDULE_ARR
------------ ------------ -------------------- --------------------
           1            4 01/01/01 12:05PM      01/01/01 3:00PM


CS342 SQL> ;
  1  select unique t.TrainNumCode, s.*
  2  from djm_Train t inner join djm_Assigned_To a on (t.TrainNumCode=a.TrainNumCode)
  3     inner join djm_Schedule s on (a.ScheduleCode=s.ScheduleCode
  4       and s.Schedule_Dept <='01/01/01 3:00PM'
  5*      and s.Schedule_Arr >'01/01/01 12:00PM')
```

Query 6 – List all cars pulled by TrainNum 1

```
CS342 SQL> @q6.sql

CARNUMCODE TRAINNUMCODE CARSTARTD CARENDDAT
---------- ------------ --------- ---------
         1            1 01-JAN-01
         2            1 02-JAN-01


CS342 SQL> ;
  1  select unique c.CarNumCode, p.TrainNumCode, c.CarStartDate, c.CarEndDate
  2  from djm_Cars c inner join djm_Pulls p on (c.CarNumCode=p.CarNumCode
  3*    and TrainNumCode = 1)
```

Query 7 – List all working Trains

```
CS342 SQL> @q7.sql

TRAINNUMCODE TRAINSTAR TRAINENDD
------------ --------- ---------
           4 04-FEB-01
           5 19-MAR-03
           7 04-APR-04
           8 12-DEC-01
          10 23-JUL-09

CS342 SQL> ;
  1  select t.* from djm_Train t
  2  MINUS
  3  select t2.* from djm_Train t2
  4* where t2.TrainEndDate <='01-January-9999'
```

Query 8 – List all working Cars

```
CS342 SQL> @q8.sql

CARNUMCODE   NUMSEATS CARSTARTD CARENDDAT
---------- ---------- --------- ---------
         1         40 01-JAN-01
         2         40 02-JAN-01
        11         40 02-JAN-02
        26         40 23-JUL-01

CS342 SQL> ;
  1  select c.* from djm_Cars c
  2  MINUS
  3  select c2.* from djm_Cars c2
  4* where c2.CarEndDate <= '01-January-9999'
```

Query 9 – List the cities that TrainNum 1 travel to

```
CS342 SQL> @q9.sql

STATIONNAME     STATIONPHON STATIONADDRESS            STATIONCODE
--------------- ----------- ------------------------- -----------
Santa Anna      6667775656  56 5th St. Santa Anna, CA           5
Los Angeles     6669996666  6th St. Los Angeles, CA             6

CS342 SQL> ;
  1  select unique s.*
  2  from djm_TrainStation s inner join djm_EnRoute_To e
  3       on (s.StationCode=e.StationCode)
  4     inner join djm_Assigned_To a on (a.TrainNumCode = 1
  5*      and a.ScheduleCode = e.ScheduleCode)
```

Query 10 – List the securityGuards assigned to TrainNum 2

```
CS342 SQL> @q10.sql

   EMPCODE EMPFNAME        EMPLNAME        EMPPHONENUM
---------- --------------- --------------- ------------
         2 Paul            Lap             7886564434

CS342 SQL> ;
  1  select unique e.*
  2  from djm_Employee e inner join djm_SecurityGuard s on (e.EmpCode = s.SGCode)
  3     inner join djm_Assigned_To a on (s.SGCode = a.EmpCode
  4*      and a.TrainNumCode = 2)
```

Phase IV

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

Procedural Language/Structured Query Language, PL/SQL, developed by Oracle and Microsoft's Transaction-Structured Query Language, MS Trans-SQL, are similar despite their syntax. Both languages support transaction processing for a database. These would be control statements, declaration of variables, error outputs, and functions that call for date and time.

The purpose of a stored subprogram is handiness and accessibility. A subprogram is a group of SQL statements written and saved to be called upon later. Instead of repeatedly entering SQL statements, subprograms may be called which promotes re-usability, maintainability, and performance. Internal details of a subprogram may be altered instead of changing other subprograms that invoke it.

2) Oracle PL/SQL

Procedural Language/Structured Query Language is structured by blocks. In each statement, there are three blocks with the first block section being declaration. This begins with DECLARE in which variables, cursors, constants, and exceptions are defined. The second block is the executable statement in between the words BEGIN and END. Finally, the third block is used for exceptions and the outcome of the exceptions.

Exception Handling- PL/SQL allows for users to use conditional control statements which include "if, then, else" statements. The syntax for such is as follows.

- IF condition THEN

  statements;

  END IF;

- IF condition THEN

  statements;

  ELSE

  statements;

  END IF;

- IF condition THEN

  statements;

  ELSEIF condition THEN

  statements;

  ELSE

  statements;

  ENDIF;

Loops

The next control statements is a loop control which behaves as a for loop, a while loop, or a goto statement.

- LOOP

  statements;

  EXIT WHEN condition;

  END LOOP;

- FOR variable IN lowerbound..upperbound LOOP

  statements;

  END LOOP;

  Cursors in SQL are statements that traverses the row of a table

- FOR cursor_variable IN cursor_name LOOP

  statements;

  END LOOP;

- WHILE condition LOOP

  statements;

  END LOOP;

- GOTO label_name

  …

  <<label_name>>

Stored Procedures

Stored procedures are precompiled procedures that are saved in the database. Stored procedures also reduce client/ server traffic and are efficient and reusable because SQL server did not have to compile an execution plan completely. SQL server only had to finish optimizing the stored plan for the procedure. The syntax is as follows:

CREATE PROCEDURE name [(parameter[, parameter,…])] IS

[local declarations]

BEGIN

executable statements

[EXCEPTION

```
        exception handlers]

    END [name];
```

Stored Functions

Users can write user-defined functions in PL/SQL or JAVA to provide functionality that is not available in SQL. User-defined functions can appear in a SQL statement anywhere SQL functions can appear, which is wherever an expression can occur. The difference between a stored procedure and stored function is that a function returns a variable. The syntax is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name
[ (variablename        IN|OUT        variabletype)]
RETURN datatype;
AS

(DECLARE variables go here)
BEGIN
        SQL statements;
        RETURN variable;
END;
```

PL/SQL Package

A package is a group of stored procedures, subprograms, stored functions where multiple procedures may be called upon.

```
CREATE PACKAGE package_name AS
        PROCEDURE names..;
        FUNCTION names…;

END package_name;

CREATE PACKAGE BODY package_name AS
        PROCEDURE name IS…
        BEGIN
                Statements
        END;

        FUNCTION name RETURN DATATYPE IS…
        BEGIN

                Statements

                RETURN variable
        END;

END package_name;
```

Triggers

Triggers provide a way of executing PL/SQL code automatically by a specific occurrence in the database, such as update, insert, delete. When such procedures are called, a trigger may be set to execute code which may record such events, like a log file.

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

3) Oracle PL/SQL Subprograms

djm_topNAvgSalary -This stored procedure returns the top n avg salary of security guards

```
--select djm_topNAvgSalary(3) from dual
CREATE OR REPLACE FUNCTION djm_topNAvgSalary( n IN NUMBER) RETURN NUMBER IS
    s NUMBER(9,2) := 0.0;
    p NUMBER(7,2) ;
    CURSOR c1 IS SELECT SGSalary FROM djm_SecurityGuard
        ORDER BY SGSalary DESC;
BEGIN
    open c1;
    FOR i IN 1..n LOOP
        fetch c1 into p;
        s := s+p;
    END LOOP;
    CLOSE c1;
    RETURN s/n;
EXCEPTION
    when others then
     raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
                '-ERROR-' || SQLERRM );
END djm_topNAvgSalary;
/
```

djm_insertEmployee – This procedure inserts an employee

```
CREATE OR REPLACE PROCEDURE djm_insertEmployee(

    EmpCode IN number,
```

```
    EmpfName IN varchar2,
    EmplName IN varchar2,
    EmpPhoneNum IN varchar2)
    AS
    BEGIN
        insert into djm_Employee values(
            EmpCode,
            EmpfName,
            EmplName,
            EmpPhoneNum);
    EXCEPTION
    when others then
     raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
                '-ERROR-' || SQLERRM );
    END djm_insertEmployee;
/
```

djm_deleteEmployee – This procedure deletes a record of employee

```
CREATE OR REPLACE PROCEDURE djm_deleteEmployee(Emp_Code IN number)
    AS
    BEGIN
        delete from djm_Employee
        where EmpCode = Emp_Code;
    EXCEPTION
    when others then
     raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
                '-ERROR-' || SQLERRM );
    END djm_deleteEmployee;
/
```

Trigger – This trigger is set off for the occurrence of update security guard

```
--update djm_SecurityGuard set SGSalary = 450 where SGCode = 1;
CREATE OR REPLACE TRIGGER djm_salary_afterUpdate
after update of SGSalary on djm_SecurityGuard
for each row
BEGIN
    insert into djm_SalaryLog
    values(salary_log_sequence.nextval, sysdate, :old.SGCode, :old.SGAddress, :old.SGSalary, :new.SGSalary);
END;
/
```

Salary Log Sequence

```
CREATE SEQUENCE salary_log_sequence
START WITH 1
INCREMENT BY 1
CACHE   3
/
```

Table for Salary Log

```
create table djm_SalaryLog(
    logNo   number NOT NULL primary key,
    eventdate date NOT NULL,
    SGCode number(5) NOT NULL,
    SGAddress varchar2(45) NOT NULL,
    SGSalary number(5) NOT NULL,
    new_SGSalary number(5) NOT NULL
);
```

Phase V

1) **Daily Activities**

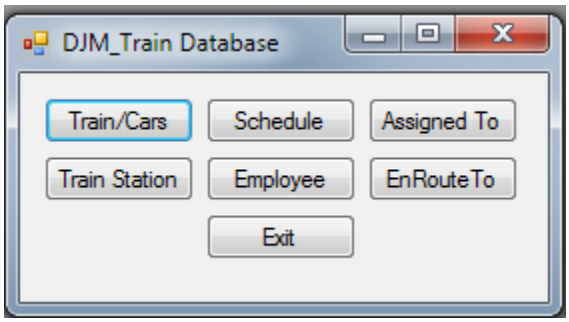
Users of the database that do not necessarily need access to all the data would be employees. Employees would use a generated schedule to see what train and time they are scheduled for. Once the employees have looked up their schedule, they will report to their duties and carry on with their job and duties.

Other users of the database would be owners of the company or bosses of the employees in which database management users could schedule employees, set up train schedules, assign employees and trains to schedules, rotate cars or trains if maintenance or tune ups were required, etc. All of which for each record can be updated, inserted, or deleted.

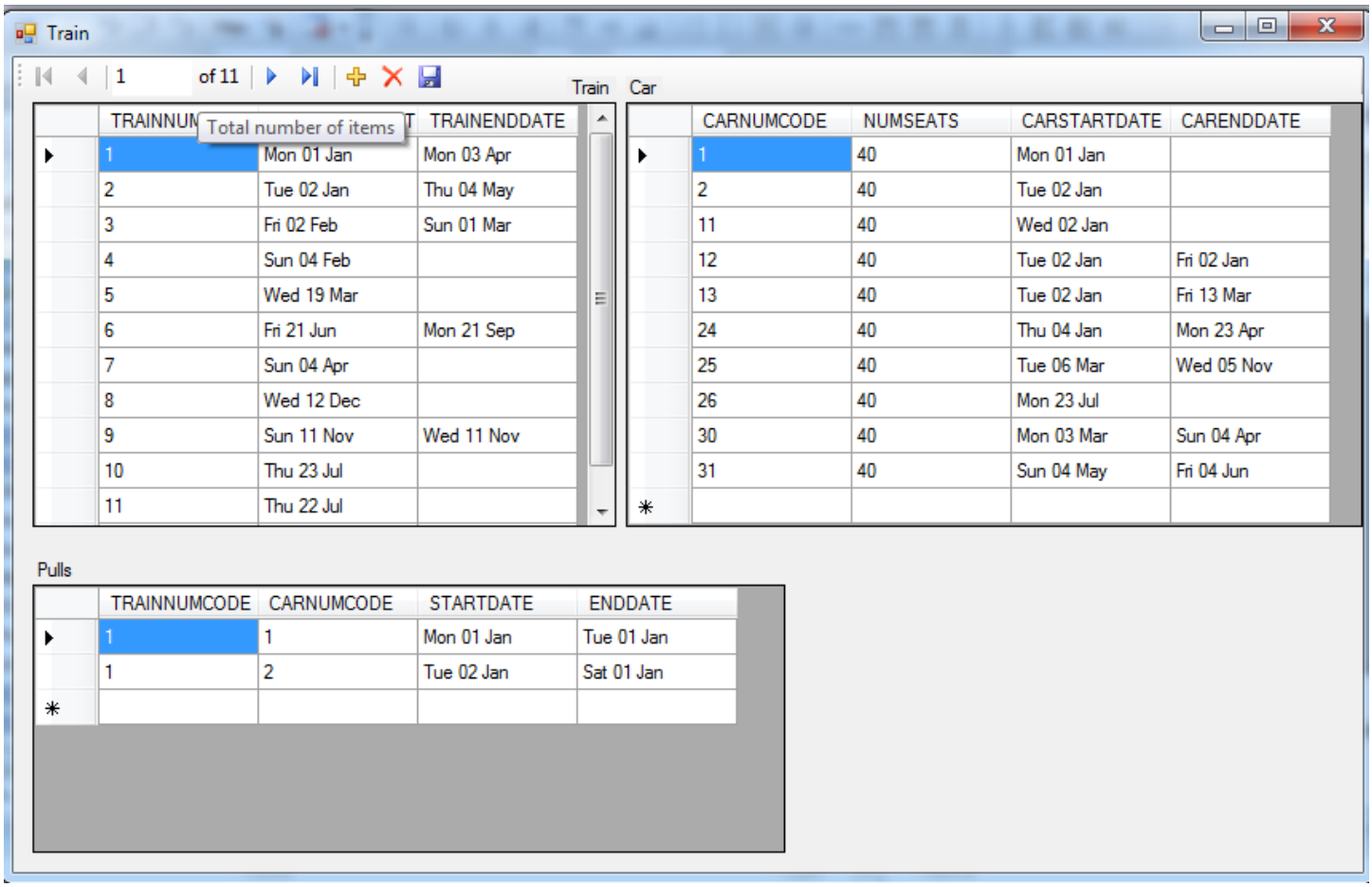
2) **Relations, Views, and Subprograms**

In order for the application to retrieve data, access was needed to the database, in which a TableAdapter for each table was created. TableAdapter provide communication between the application and the database. Also, a TableAdapter can execute queries or stored procedures, and either return existing data or send new updated data from the application to the database. Each table had and individual TableAdapter: djm_Assigned_ToTableAdapter, djm_CarsTableAdapter, djm_EmployeeTableAdapter, djm_EngineerTableAdapter, djm_Enroute_ToTableAdapter, djm_PullsTableAdapter, djm_SalaryLogTableAdapter, djm_ScheduleTableAdapter, djm_SecurityGuardTableAdapter, djm_TrainTableAdapter, djm_TrainStationTableAdapter. By acquiring the TableAdapter, a DataGridView was created to see the existing data for each table used in the application.

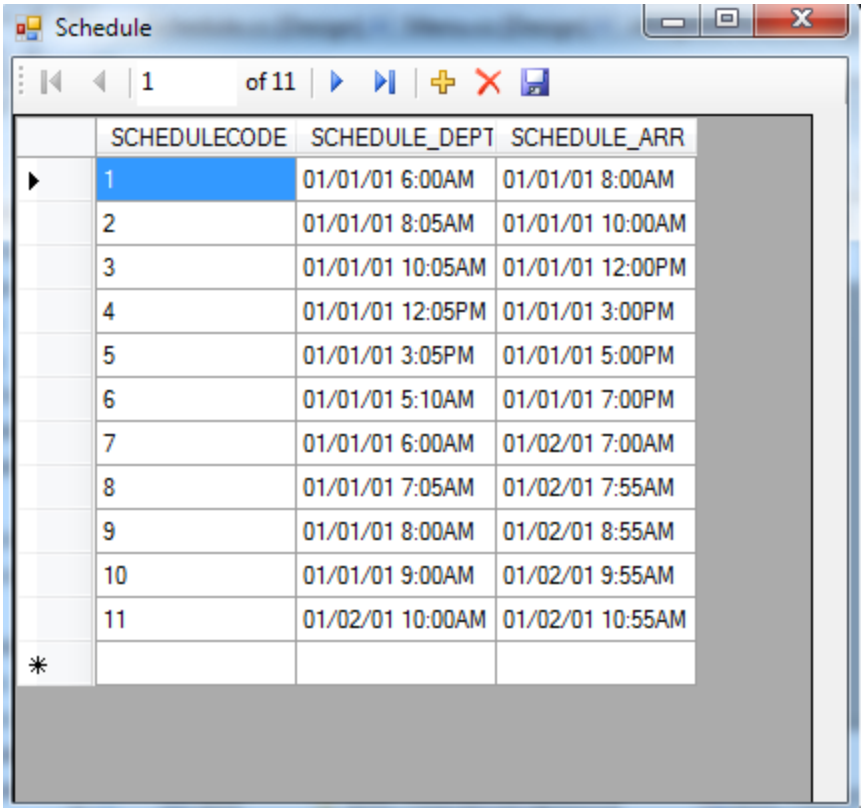
3) **Screenshots**



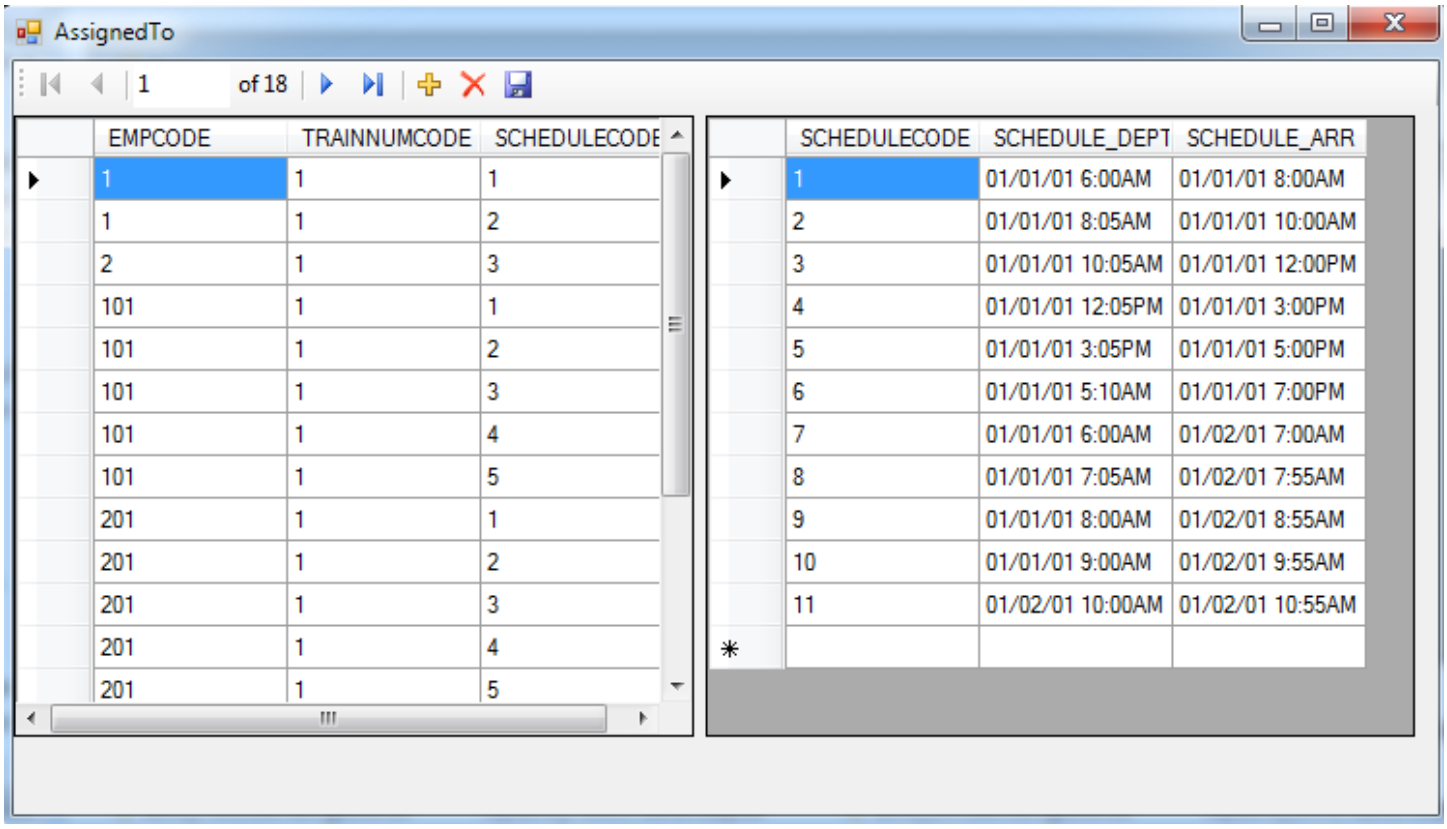
This is the Main Interface when the application is started. There are several buttons in which the user can access tables.



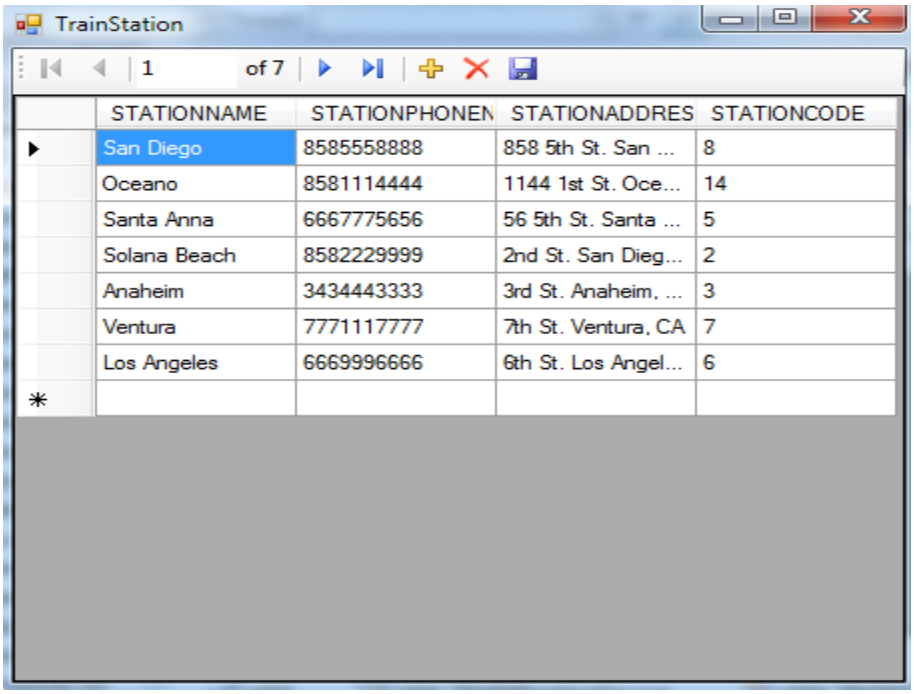
The Train Button from the Menu will open up a new window form called Train. This window shows dataGridViews of Train, Cars, and the relationship 'Pulls' between the two. Also, navigation buttons are located at the top along with insert, delete, and save. In addition, any record may be edited simply by clicking on a cell and changing the data inside.



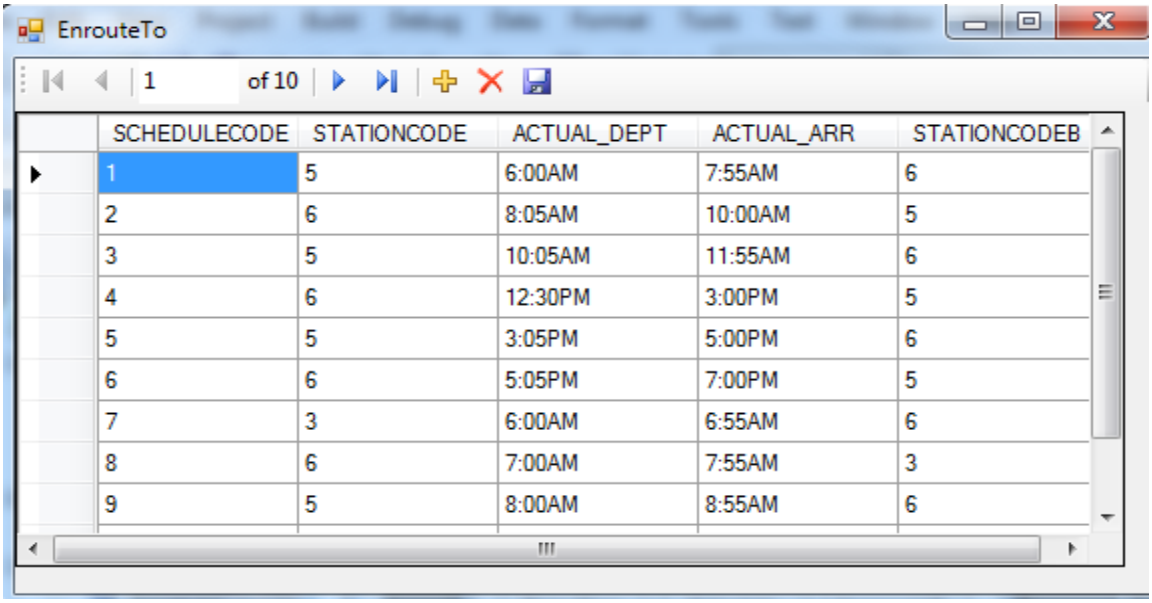
When the Schedule button is pressed in the Menu window, a new window form called Schedule is opened. This shows a dataGridView of the table djm_Schedule. Also, navigation buttons are located at the top along with insert, delete, and save. In addition, any record may be edited simply by clicking on a cell and changing the data inside.



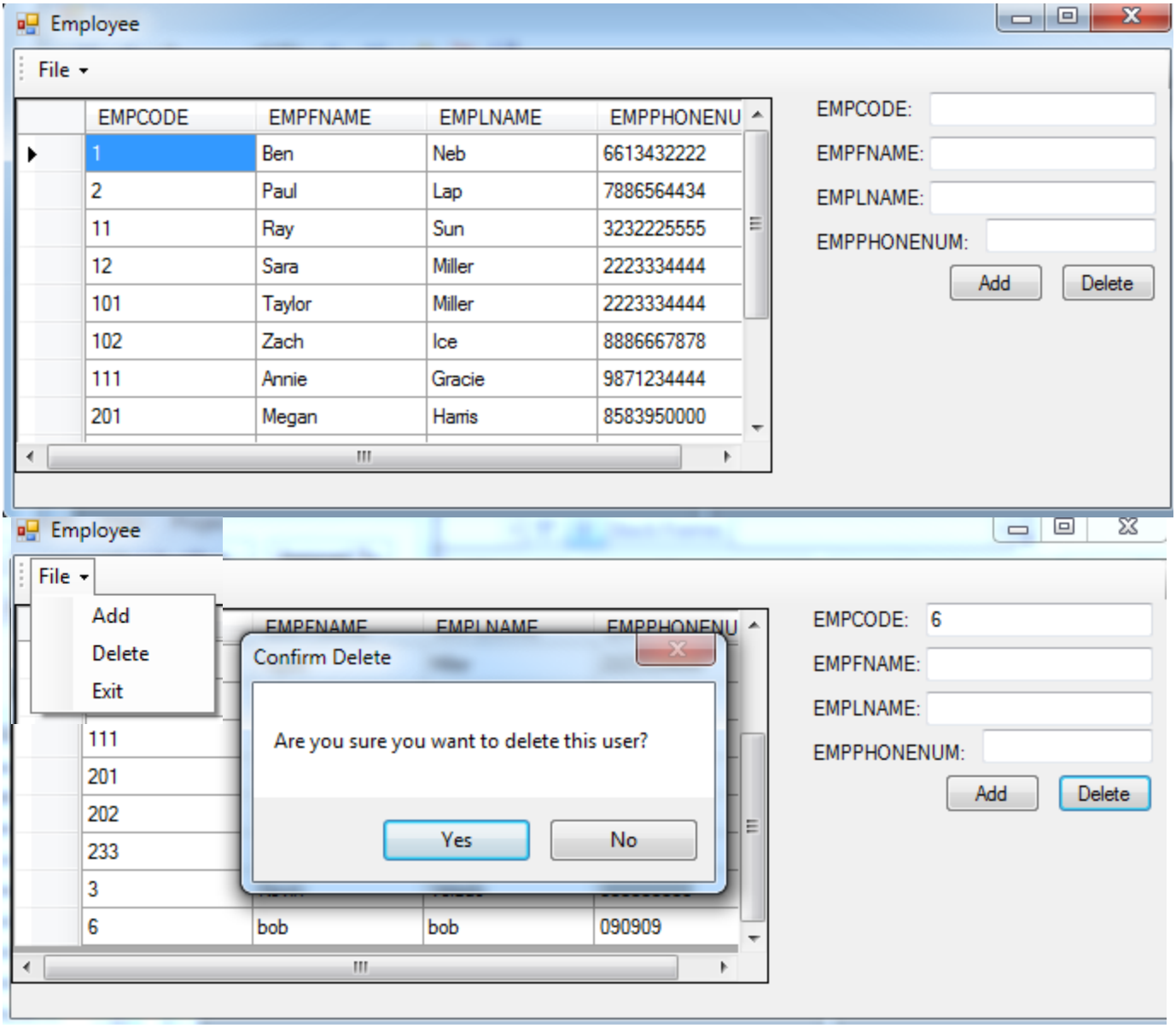
When the Assigned To button is pressed from the Menu window, a new window form is opened called Assigned To. This window shows dataGridViews for two tables, djm_Assigned_To, and djm_Schedule. Most employee users would use this part of the application to look up their schedule and which train they are assigned to. Also, navigation buttons are located at the top along with insert, delete, and save. In addition, any record may be edited simply by clicking on a cell and changing the data inside.



When the Train Station button is pressed in the Menu Window, a new window form called Train Station is opened. In this window, a dataGridView of existing data in djm_TrianStation is shown. Also, navigation buttons are located at the top along with insert, delete, and save. In addition, any record may be edited simply by clicking on a cell and changing the data inside.



When the EnRoute To button is pressed in the Menu Window, a new window form called EnrouteTo is opened. This window displays a dataGridView of djm_EnRoute_To. Also, navigation buttons are located at the top along with insert, delete, and save. In addition, any record may be edited simply by clicking on a cell and changing the data inside.



When the Employee button is pressed in the Menu Window, a new window form is open called Employee. This displays a dataGridView of djm_Employee. Two more buttons are also included, Add and Delete, along with a top lined tool bar wth drop down menu with, Add, Delete, and Exit. This window also confirms a deletion of a record with a yes or no confirmation.

4) **Description of GUI Code**

Major steps of designing a user interface – The first initial step was to create a new project in Visual Studios 2008 using Visual C# and an installed template called Windows Form Application. Also, the application used .NET Framework 3.5. Once this was created, I inserted a Windows Form which is a template of a window that may hold a variety of objects. This is the basis for each window in the application. Visual Studios has a "Toolbox" which contains common controls such as a button, data containers such as dataGridViews, menu options, text boxes, labels, etc. All of which are as easy as drag and drop and code is automatically written in the background.

The next step was setting up the connection to the database helios.cs.csubak.edu. In Visual Studios 2008, a feature called "add new data source" was essentially a wizard for setting up this connection. In order to use this feature for Oracle, the Oracle client had to be installed. By using the net manager, a connection to helios was established in which I was able to select which tables, views, subprograms, packages, etc. I wanted to add as a data source. Once the database was added to Visual Studios, it was a drag and drop into a window form to display a dataGridView. By doing this, TabaleAdapters, BindingSources, openConnection, and closeConnection were automatically coded.

Connecting to Helios

```
//DB IP address as well as Login, and Password
                string oradb = "Data Source=(DESCRIPTION=" +
"(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=helios.cs.csubak.edu)(PORT=1521)))" +
"(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=ORCL)));" + "User
Id=cs342;Password=c3m4p2s;";

//Make an Oracle Connection Object, and Assign the DB Login Info to it
                OracleConnection conn = new OracleConnection(oradb);
                conn.ConnectionString = oradb;

//Open the Oracle Connection
                conn.Open();
```

(Code examples from Employee will be shown because every window application used Employee like a template)

Fill Data – Established a connection and filled the data into a dataGridView

```
void FillData()
        {
            using (OracleConnection c = new
OracleConnection(Properties.Settings.Default.ConnectionString))
            {
                c.Open();
                using (OracleDataAdapter a = new OracleDataAdapter("SELECT * FROM
djm_Employee", c))
                {
                    DataTable t = new DataTable();
                    a.Fill(t);
                    dJM_EMPLOYEEDataGridView.DataSource = t;
                }
            }}
```

Refresh Employee – Whenever an update, insert, delete command is made, the table must be refreshed

```
private void refreshEmployee()
        {
            dJM.DJM_EMPLOYEE.Clear();
            FillData();
        }
```

Add Employee – Inserts an employee into the table and refreshes the table

```
private void AddEmp_Click(object sender, EventArgs e)
        {
            OCommand cEntry = new OCommand();
            string ecode = this.txtecode.Text;
            string efname = this.txtefname.Text;
            string elname = this.txtelname.Text;
            string ephone = this.txtephone.Text;
            string command = "insert into djm_Employee values
("+ecode+",'"+efname+"','"+elname+"','"+ephone+"')";
            cEntry.Run(command);
            refreshEmployee();
        }
```

Delete Employee – Deletes an employee record, confirms action with a yes or no, and refreshes the table

```
private void DelEmp_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("Are you sure you want to delete this user?", "Confirm
Delete", MessageBoxButtons.YesNo)
                == DialogResult.Yes)
            {
                OCommand cEntry = new OCommand();
                string ecode = this.txtecode.Text;
                string command = "delete from djm_Employee where EmpCode=" + ecode;
                cEntry.Run(command);
            }
            refreshEmployee();
        }
```

Major Features of GUI - The major feature of this application is to look up important records from the database by using a simple graphical user interface. Without the graphical user interface, a command prompt like window would be used along with manually entering long and easily forgetful sql commands. By using the GUI, it allows for easy usability, higher productivity, greater accessibility, lower cognitive lode, and higher productivity for the user or organization.

Development and Learning - The best way I learned about using Visual Studios, C#, Oracle, and any other type of algorithms was the internet and forums. The internet has offered a social network between people and companies that came about similar situations as I did during the creation of this application. I discovered many solutions to one situation in which some solutions were simple and some were a degree of high complexity.

5) **Designing and Implementing a Database Application**

My design for this database application was quite simple. I liked the idea of a menu window and branching off to different characteristics of the database. So, from the main menu, I could open the Employee part of the database by clicking a button. From there, a new window would open up with every record in a data table along with a couple of function buttons, insert and delete. In addition to buttons, I included a drop down menu with the same functions along with exit, which closes the window. The basis for my database application is essentially to look up records, insert, delete, and update records in the database. If I had more time, I would like to add more detailed attributes and records into the database. Since the database only goes to a degree of scheduling employees and trains, I would like to add the business side to the organization. This would include passengers, sales, luggage, and maybe even ticket packages which may include bus services, car rentals, hotel stays, and dining. I have learned that a database can contain information to a near endless degree and can be difficult to maintain. In order to maintain the database, the creator must design the database in the most simplified form. This begins with data modeling. I actually went back to my original ER diagram and made some alterations. One alteration was getting rid of unrelated entities to the database. Another alteration I made was to keep the names and phone numbers in the Employee entity only, rather than both, the parent class (Employee) and the subclasses (Security Guard, Conductor, Engineer).

6) **Conclusion**

Even though this project is small and simple, it is a great example to real-world application of a database. This project is a great starting foundation in the field of managing a database. This class was a great experience and has pushed me to strive for knowledge. Topics that I learned are as followed: what a database is, database modeling (entities, attributes, constraints), relational algebra, relational calculus, sql commands, creating tables into the database, stored procedures, implementing the database into visual studios with C#, and creating a graphical user interface (which is something that I've never done before).