City of Delano
Public Works Department
Sewer & Waste Treatment
Water Production

FINAL PROJECT

Eddie Rangel
CMPS 342 – Database Systems
Dr. Huaqing Wang
November 24, 2010

Table of Contents

I. Fact-Finding, Information Gathering and Conceptual Database Design 5

a. Fact Finding Techniques and Information Gathering 5

i. Methods Used 5

ii. Introduction to the Organization 5

iii. Structure of the Organization 5

iv. Major Objects 6

v. Data Views 6

b. Conceptual Database Design 6

i. Entity Set Description 6

ii. Relationship Set Description 9

iii. Related Entity Set 9

iv. E-R Diagram 9

II. From ER (Conceptual) Model to Relational (Logical) Model 10

a. ER Model and Relational Model 10

i. Description 10

ii. Comparison 10

iii. Conversion 11

iv. Define Constraints 11

b. Convert the ER Database 11

i. EMPLOLYEE Relation 11

ii. DEPARTMENT Relation 12

iii. SERVICE_REQUEST relation 12

iv. TECHNICIAN relation 13

v. TYPE relation 13

vi. ASSET relation 14

vii. WORKS_FOR relation 14

viii. WORKS_ON 14

ix. HAS_A 15

x. SUMBITS_A 15

xi. IS_A 16

xii. POSSESES 16

xiii. SKILL 16
c. Design Relation Instances 17
i. EMPLOYEE(empID, fName, lName, mName, DOB, SSN, PhoneNumber) 17
ii. DEPARTMENT(deptID, deptName, Location) 17
iii. SERVICE_REQUEST(serviced,openDate, closeDate) 17
iv. TECHNICIAN(empID, skillID) 17
v. TYPE(typeID, aType) 18
vi. ASSET(assetID, InsDate, LstMaintenance, Location) 18
vii. WORKS_FOR(deptID , strDate, endDate) 18
viii. WORKS_ON(serviceID, empID, strDate, endDate) 18
ix. IS_A(typeID,assetID) 18
x. HAS_A(serviceID, typeID) 18
xi. SUBMITS_A(empID, serviceID) 19
xii. POSSESES(empID,skillID) 19
xiii. SKILL(skillID, Name, Description) 19
d. Queries 19
e. Query Representation 20
III. Implementation of Relational Database 23
a. SQL*PLUS 23
b. Schema Objects in Oracle 23
i. Tables 23
ii. Views 23
iii. Dimensions 24
iv. Sequences 24
v. Synonyms 24
vi. Indexes 24
vii. Database Links 24
viii. Stored procedures and functions 24
ix. Packages 24
c. Schema Objects in this project 24
IV. Stored Subprograms, Packages and Triggers 35
a. Common Features in Oracle PL/SQL and Microsoft Transact-SQL 35

b. Oracle PL/SQL 35
c. Oracle PL/SQL Subprograms 38
V. Graphical User Interface Design and Implementation 41
a. Daily Activities 41
i. Administrative Staff 41
ii. Technicians 41
iii. Managers 41
The Management Team can add/delete all items in the database. These items include employees, assets, and service requests. They can keep track of their technicians certifications. 41
b. Relations, Views and Subprograms 41
i. RELATIONS 41
ii. VIEWS 41
iii. SUBPROGRAMS 41
c. Screen Shots 42
d. Description of Code 47
e. Major Steps in Design and Implementation 49

I. Fact-Finding, Information Gathering and Conceptual Database Design

a. Fact Finding Techniques and Information Gathering

Fact finding is used to gain an understanding of what will be needed in order to develop and implement an Information Management System. There are a variety of ways that information can be gathered including phone conversations and e-mails consisting of survey as well as face to face meetings.

The information you are seeking will generally direct you to which methods you will use. If you are looking for a generic response from many people then surveys will be a very convenient way to go. However, if you would like a very detailed account of specific requirements then face to face meetings will be more appropriate.

Once you begin understanding what the requirements will be you can begin to develop small components that will eventually make up the final System.

i. Methods Used

- Kick off meeting
  The Development Team made arrangements to have an initial face to face meeting with some of the stakeholders of the project. The Stakeholders consisted of Department Heads and Middle Managers of the divisions that will work directly with the System once it is in place.

- Interviews
  Following the initial meeting the Development Team began making individual one on one meetings with many of the Divisions staff members. This included the Administrative Secretary and Office Assistant to the Division Manager and Lead Technicians.

  The Team began documenting the internal work flow for a service request for the Water and Sewer Divisions. Copies of their forms were obtained to better understand what information the Division wanted to keep track of.

ii. Introduction to the Organization

The City of Delano's population has reached more than 53,972, as of 2009. This makes it the second largest City by population in Kern County. The City operates many essential services for the Citizens of Delano including Water Utilities, Solid Waste, and Streets.

The City's current System for managing the day to day operations of the Public Works Department is outdated and tedious. Many of their routine tasks are done manually and on hard copies. They would like to stream line common tasks into an automated system. This will help manage their Work Orders and will provide a way track Citizen Requests.

iii. Structure of the Organization

This project will focus specifically on the Water Production and Sewer Divisions of the Public Works Department. The Divisions consists Certified Technicians that manage the Cities

Infrastructure. The Department will enter a service request when they need to service one of their assets which include water pipes, water valves, and fire hydrants.

iv. Major Objects

The Database System will use an Entity named EMPLOYEE to represent the standard employee for the City. Any EMPLOYEE can work for only one DEPARTMENT. There are certain EMPLOYEEs who HAS_SKILLS that are TECHNICANs. The TECHNCIAN WORKS_ON submitted SERVICE_REQUESTs for ATYPE of asset. ATYPE of asset is defined by the attributes within the entity ASSET.

v. Data Views

Each employee can view their SERVICE_REQUESTs. Due to the complexity of some of the SERVICE_REQUESTs multiple TECHNICIANs may work on a single SERVICE_REQUEST. Only one type of asset can be related to a SERVICE_REQUEST.

b. Conceptual Database Design

i. Entity Set Description

- EMPLOYEE
  - This Entity Type represents all Employees in the City. Since this database will focus on one Department, we will only require basic employee information to be used.
  - Candidate keys: empID, SSN
  - Primary key: empID
  - Strong/Weak Entity: Strong
  - Fields to be indexed: empID

Attribute/ Description	**empID**	**fName**	**lName**	**mName**	**DOB**	**SSN**	**Phone Number**	**Wage**
Description	Unique Identifier	First Name	Last Name	Middle Name	Date of Birth	Social Security Number	Home Number	Hourly Rate
Domain/ Type	Char	Varchar	Varchar	Varchar	Date	nchar	Nchar	float
Value/ Range	9	50	50	50		9	10	
Default Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
NULL (Y/N)	N	N	N	Y	N	N	Y	N
Single/ Multi	Single	Single	Single	Single	Single	Single	Multi	Single
Simple/ Composite	Simple	Simple	Composite	Simple	Simple	Simple	Composite	Simple

- DEPARTMENT
  - This Entity Type represents all Departments in the City.
  - Candidate keys: deptID
  - Primary key: deptID
  - Strong/Weak Entity: strong
  - Fields to be indexed: deptID

Attribute/ Description	**deptID**	**deptName**	**deptLocation**
Description	Unique Identifier	First Name	Last Name
Domain/ Type	Char	Varchar	Varchar
Value/ Range	9	50	50
Default Value	N/A	N/A	N/A
NULL (Y/N)	N	N	N
Single/ Multi	Single	Single	Single
Simple/ Composite	Simple	Simple	Composite

- TYPE
  - This entity represents a type of asset being serviced.
  - Candidate keys: assetID
  - Primary key: assetID
  - Strong/Weak Entity: Strong
  - Fields to be indexed: assetID

Attribute/ Description	**typeID**	**aType**
Description	Unique Identifier	First Name
Domain/ Type	Char	Varchar
Value/ Range	9	50
Default Value	N/A	N/A
NULL (Y/N)	N	N
Single/ Multi	Single	Multi
Simple/ Composite	Simple	Simple

- ASSET
  - Each type of asset is defined by a set of attributes with the ASSET entity
  - Candidate keys: assetID
  - Primary key: assetID
  - Strong/Weak Entity: Strong
  - Fields to be indexed: assetID

Attribute/ Description	**assetID**	**Geometry**	**InsDate**	**LstMaintenance**
Description	Unique Identifier	First Name	Last Name	Middle Name
Domain/ Type	Char	varchar	DateTime	DateTime
Value/ Range	9	10		
Default Value	N/A	N/A	N/A	N/A
NULL (Y/N)	N	Y	N	N
Single/ Multi	Single	Single	Single	Single
Simple/ Composite	Simple	Simple	Simple	Simple

- SERVICE_REQUEST
  - The entity SERVICE_REQUEST will be used to track when a SERVICE_REQUEST is opened and closed.
  - Candidate keys: serviceID
  - Primary key: serviceID
  - Strong/Weak Entity: Strong
  - Fields to be indexed: assetID

Attribute/ Description	**serviceID**	**openDate**	**closeDate**
Description	Unique Identifier	Open Date	Close Date
Domain/ Type	Char	DateTime	DateTime
Value/ Range	9		
Default Value	N/A	N/A	N/A
NULL (Y/N)	N	N	N
Single/ Multi	Single	Single	Single
Simple/ Composite	Simple	Simple	Simple

ii. Relationship Set Description

- WORKS_FOR
  - This is a Relationship between and EMPLOYEE and DEPARTMENT
  - Entity Set Involved: EMPLOYEE, DEPARTMENT
  - Mapping cardinality: 1..1
  - Participation Constraints: Total Participation for EMPLOYEE

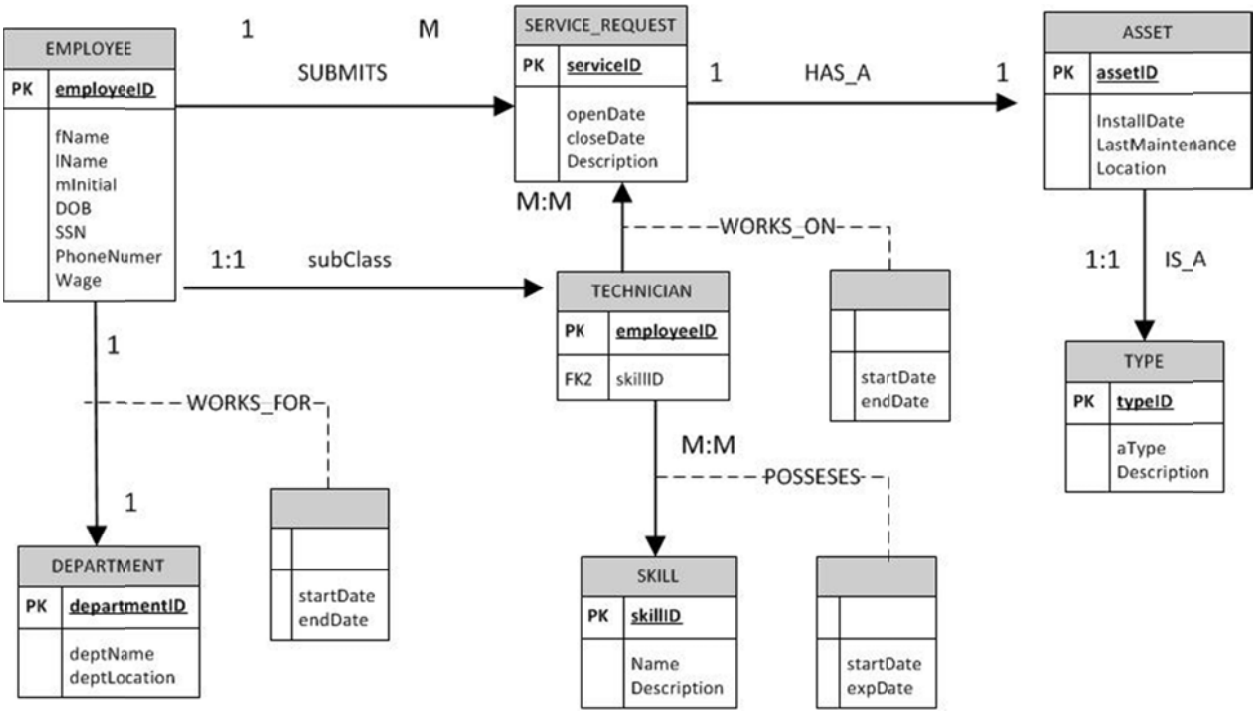
- POSSESES
  - This is a Relationship between and EMPLOYEE and TECHNCIAN
  - Entity Set Involved: EMPLOYEE, TECHNICIAN
  - Mapping cardinality: M..M
  - Participation Constraints: This is an optional relationship for EMPLOYEE

- WORKS_ON
  - This is a Relationship between TECHNCIAN and SERVICE_REQUEST
  - Entity Set Involved: TECHNICIAN, SERVICE_REQUEST
  - Mapping cardinality: M..M
  - Participation Constraints: Total Participation for TECHNICIAN

iii. Related Entity Set

- TECHNICIAN
  - TECHNICIAN is a specialization of EMPLOYEE. It is disjointedness constraint.
  - Specialization/Generalization
  - Aggregation/has-relationship

iv. E-R Diagram



II. From ER (Conceptual) Model to Relational (Logical) Model

a. ER Model and Relational Model

i. Description

Prior to the implementation of a database system there are some prerequisite steps that need to be completed. These steps include the creation of an Entity-Relationship Model and the Relational model. The ER Diagram previously discussed is a representation of the organization of the systems data. It serves as a guide in determining the required components of a developed database system.

The creation of a relational model is required in order to develop an efficient application for the organization. It is defined by a set of relations which have constraints on their specific domains.

ii. Comparison

The entity relationship is meant to provide a framework of information for which to base a conceptual model on. It does not require specific details required for an implementation. Instead it is focused on providing a visual representation of how data is organized and where it flows. It provides attributes, relationships, and cardinality.

The relational model provides the database developer with an understanding of how the data will be constrained for it given domains. It builds on the ER model by creating valid tuples and their given relationships. This model is a more accurate representation of how the database will physically be created.

iii. Conversion

There are several steps to take when converting an ER model to a relational model. A conceptual design provides the basic structure from which the relational model will be built and the relational model's relations will be mapped.

The first step will include creating a relation R for every regular strong entity type. This includes all simple attributes for the entity E. A primary key will be chosen from the attributes of the entity to uniquely identify the relation R.

The second step will include the mapping of the weak entity types. For each week entity type we will create a relation R and will include all the simple attributes for the entity E. We will also include a foreign key attribute.

The third step will map binary 1:1 relations types. For every binary relation R we need to identify its relations as S and T. There are three possible approaches which include the Foreign Key method, the merged relation method, and the cross-reference method.

In the fourth step we will focus on mapping the binary 1:M relationship types. For every 1:M relation R we will identify the S that corresponds the entity of the M-side of the relationship. We include a foreign key in S the primary key of the relation T.

A fifth step includes the mapping of binary M:N relationship types. For every binary M:N relation R we will create a new relation S to represent R. We will include a foreign key attribute in S.

iv. Define Constraints

A relation consists of an ordered set of unique tuples, with each tuple having the same amount and type of attributes. In the relational model Entities are represented and each row is a valid instance, record, or tuple for the entity. The constraints ensure that no primary key can be NULL. This ensures that there exists a unique element of each tuple in the relation. This is necessary for comparisons and representations in queries and data integrity. The constraints for foreign keys exist to enforce referential integrity.

b. Convert the ER Database
   i. EMPLOLYEE Relation
      - Attributes
        - empID
          Domain: Integer, Cannot be NULL
        - fName
          Domain: String, up to 30 Characters
        - lName
          Domain: String variable characters, cannot be NULL

- mName
  Domain: String variable characters, can be NULL

- DOB
  Domain: DateTime, Cannot be Null

- SSN
  Domain: Integer, Cannot be Null

- Wage
  Domain: Currency, cannot be NULL

- PhoneNumber
  Domain: String, cannot be NULL

- Constraints
  - Primary Key
    empID will be the primary key. This unique identifier cannot be NULL.
  - Foreign Key
    Works_For is the Foreign Key which corresponds to a deptID. All employees belong to one department.
  - Business Rules
- Candidate Keys
  - empID, SSN

ii. DEPARTMENT Relation

- Attributes
  - deptID
    Domain: Integer, Cannot be NULL
  - deptName
    Domain: String, up to 30 Characters, cannot be NULL
  - Location
    Domain: String, up to 20 Characters, cannot be NULL

- Constraints
  - Primary Key
    deptID will be the Primary Key
  - Foreign Key
  - Business Rules

- Candidate Keys
  - deptID, deptName

iii. SERVICE_REQUEST relation

- Attributes
  - serviceID: DateTime, cannot be NULL

- openDate: DateTime, cannot be NULL
- closeDate: DateTime, can be NULL

- Constraints
  - Primary Key
    serviceID is the Primary Key, it cannot be null.
  - Foreign Key
  - Business Rules
    The closeDate has to be greater than the openDate. The serviceID will be incremented.

- Candidate Keys
  - serviceID

iv. TECHNICIAN relation

- Attributes
  - empID
    Domain: integer, cannot be NULL
  - POSSESES
    Domain: integer, cannot be NULL

- Constraints
  - Primary Key
    empID
  - Foreign Key
  - Business Rules
    A TECHNICIAN is a child class of EMPLOYEE. All TECHNICIAN s must possess a skill

- Candidate Keys
  empID

v. TYPE relation

- Attributes
  - typeID:
    Domain: Integer, cannot be NULL
  - aType
    Domain: String, up to 20 Characters, cannot be NULL
  - Description:
    Domain: String, up to 150 Characters, cannot be NULL
- Constraints
  - Primary Key
    typeID is the Primary Key
  - Foreign Key
  - Business Rules

- Candidate Keys
  - typeID

vi. ASSET relation

- Attributes
  - assetID
    Domain: Integer, cannot be NULL
  - InsDate
    Domain: DateTime, cannot be NULL
  - LstMaintenance
  - Domain:DateTime, can be NULL
  - Location
    Domain: String, can be NULL
  - IS_A:
    Domain: Integer value corresponds to type of asset , cannot be NULL

- Constraints
  - Primary Key
    assetID is the unique identifier. Must only cont numeral values that increment
  - Foreign Key
    IS_A is the unique identifier that every ASSET must have that relates it to the type of asset it is.
  - Business Rules

- Candidate Keys
  - assetID

vii. WORKS_FOR relation

- Attributes
  - strDate
    Domain: DateTime, cannot be NULL
  - endDate
    Domain: DateTime, can be NULL
  - empID
    Domain: Integer, Cannot be NULL

  - deptID

- Constraints
  - Primary Key
  - Foreign Key
    empID, deptID
  - Business Rules

- Candidate Keys

viii. WORKS_ON

- Attributes
  - strDate
    Domain: DateTime, cannot be NULL

  - endDate
    Domain: DateTime, can be NULL

  - empID
    Domain: Integer, Cannot be NULL

  - serviceID
    Domain: Integer, Cannot be NULL

- Constraints
- Candidate Keys

ix. HAS_A
- Attributes
  - serviceID
    Domain: Integer, cannot be NULL

  - assetID
    Domain: Integer, cannot be NULL

- Constraints
  - Primary Key
  - Foreign Key
    serviceID, assetID
  - Business Rules

- Candidate Keys

x. SUMBITS_A
- Attributes
  - empID
    Domain: Integer, cannot be NULL

  - serviceID
    Domain: Integer, cannot be NULL

- Constraints
  - Primary Key
  - Foreign Key
    empID, serviceID
  - Business Rules

In order for a work order to close it must have an close date.

- Candidate Keys

xi. IS_A

- Attributes
  - typeID
    Domain: Integer, cannot be NULL
  - assetID
    Domain: Integer, cannot be NULL
- Constraints
  - Primary Key
  - Foreign Key

    typeID, assetID are both required
  - Business Rules
- Candidate Keys

xii. POSSESES

- Attribute
  - empID
    Domain: Integer, cannot be NULL
  - skillID
    Domain: Integer, cannot be NULL
- Constraints
  - Primary Key
  - Foreign Key
    empID & skillID are both required
  - Business Rules
    All technicians are required to have a skill. Some skills are associated with Certifications which are required to be maintained. If the Certification expires then the technician may not be able to perform certain work items.
- Candidate Keys

xiii. SKILL

- Attributes
  - skillID
    Domain: Integer, Cannot be NULL
  - SkillName
    Domain: String, Variable Length, cannot me null
  - Description
    Domain: String, Variable Length, cannot me null

- Constraints
  - Primary Key
    skillID
  - Foreign Key
  - Business Rules

- Candidate Keys
  - skillID

c. Design Relation Instances

i. EMPLOYEE(empID, fName, lName, mName, DOB, SSN, PhoneNumber)

empID,	fName	lName	mName	DOB	SSN	PhoneNumber
01	John	Rodriguez		7/1/1970	555-66-8888	661-720-2201
02	Steve	Smith		8/1/1977	555-66-9999	661-720-2202
03	Taylor	Dane		9/7/1978	555-77-6644	661-720-2202
04	Jason	Davis		10/71974	555-77-7755	661-720-2203
05	Ted	Mason		1/8/1972	555-66-8822	661-720-2204
06	Jason	Frank	Alexander	2/15/1976	555-88-6644	661-720-2205
07	Doug	Parker	Author	2/28/1978	555-88-4482	661-720-2206

ii. DEPARTMENT(deptID, deptName, Location)

deptID	deptName	Location
01	Water Production	Corporation Yard
02	Sewer	Treatment Plant
03	Waste Water	Treatment Plant
04	Water Treatment	Corporation Yard

iii. SERVICE_REQUEST(serviced,openDate, closeDate)

serviceID	srtDate	endDate	Description
01	10/10/2010		Water Leak
02	10/1/2010	10/5/2010	Water PipeMaintenance
03	9/30/2010		Clogged Drain
04	8/30/2010	9/1/2010	Waste Water Leak

iv. TECHNICIAN(empID, skillID)

empID	skillID
01	03
02	01

03	03	
04	01	
05	02	

v. TYPE(typeID, aType)

typeID	aType
01	Water Valve
02	Water Pipe
03	Waste Water Valve
04	Waste Water Pipe

vi. ASSET(assetID, InsDate, LstMaintenance, Location)

assetID	InsDate	LstMaintenance	Location
01	2/10/2000	2/10/2010	Lexington St
02	2/15/2000	2/15/2009	Lexington St
03	3/15/2005	5/30/2010	Jefferson St
04	1/30/1998	9/27/2003	Jefferson St

vii. WORKS_FOR(deptID , strDate, endDate)

empID	deptID	strDate	endDate
01	01	1/2/2000	
02	02	11/10/2005	
03	03	7/15/2008	
04	04	8/20/2003	

viii. WORKS_ON(serviceID, empID, strDate, endDate)

serviceID	empID	strDate	endDate
01	02	1/20/2010	2/1/2010
02	02	2/15/2010	2/16/2010
03	01	7/1/2010	7/2/2010
04	05	8/15/2010	

ix. IS_A(typeID,assetID)

typeID	assetID
01	01
02	02
03	03
04	04

x. HAS_A(serviceID, typeID)

serviceID	typeID
01	01
02	02

03	03
04	04

xi. SUBMITS_A(empID, serviceID)

empID	**serviceID**
01	01
02	02
03	03
04	04

xii. POSSESES(empID,skillID)

empID	**skillID**

xiii. SKILL(skillID, Name, Description)

skillID	**Name**	**Description**
01	Water Operator	State Certified
02	Waste Water Operator	State Certified
03	Maintenance Technician	Misc. Employee
04	Data Entry	Misc. Employee

d. Queries

- Select all technicians who worked on service requests.
- Select all technicians who have more than 2 service requests.
- Select All employees who have submitted a service request.
- Show what skills all technicians have
- Show Employees with no skills
- Show all open service requests
- Which employees have open service requests?
- What types of assets has service requests?
- Show Former Employees
- Show Departments with more than 1 Employee

e. Query Representation

- Select all technicians who worked on service requests.

  Relational Algebra

  π (employee_id) σ (er_technician T * er_works_on W)
  T=employee_id = W.employee_id

  Tuple Relational Calculus

  {e|er_employee(e) ^ (∃T)(er_technician(T) ^ (∃W)(er_works_on(W) ^ W.employee_id = T.employee_ID)}

  Domain Relational Calculus

```
SELECT T.EMPLOYEE_ID
FROM ER_TECHNICIAN T
WHERE EXISTS(SELECT * FROM ER_WORKS_ON W WHERE W.EMPLOYEE_ID = T.EMPLOYEE_ID);
```

- Select all technicians who have more than 2 service requests.

  Relational Algebra

  π(P.*) σ(er_possesses P)

  Tuple Relational Calculus

  {p|er_possesses(p) ^ (∃W)(er_works_on(W) ^ p.employee_id = W.employee_id) }

  Domain Relational Calculus

```
SELECT P.*
FROM ER_POSSESSES P
WHERE EXISTS(SELECT EMPLOYEE_ID, COUNT(*)
 FROM ER_WORKS_ON W
 WHERE P.EMPLOYEE_ID = W.EMPLOYEE_ID
 HAVING COUNT(*) > 1);
```

- Select All employees who have submitted a service request.

  Relational Algebra

  Tuple Relational Calculus

  {e|er_employee_id(e) ^ (∃S)(er_submits_a(S) ^s.employee_id = e.employee_ids }

Domain Relational Calculus

```
SELECT E.EMPLOYEE_ID
FROM ER_EMPLOYEE E
WHERE EXISTS (SELECT * FROM ER_SUBMITS_A S
WHERE S.EMPLOYEE_ID = E.EMPLOYEE_ID);
```

- Show what skills all technicians have

Relational Algebra

Tuple Relational Calculus

{p | er_possesses(p) }

Domain Relational Calculus

```
SELECT P.SKILL_ID, P.EMPLOYEE_ID
FROM ER_POSSESSES P;
```

- Show Employees with no skills

Relational Algebra
Tuple Relational Calculus

{e|er_employee(e) ^ ~(∃P)(er_possesses(P) ^ P.employee_id = e.employee_id}

Domain Relational Calculus

```
SELECT E.F_NAME, E.L_NAME
FROM ER_EMPLOYEE E
WHERE NOT EXISTS(SELECT * FROM ER_POSSESSES P WHERE P.EMPLOYEE_ID =
E.EMPLOYEE_ID);
```

- Show all open service requests

Relational Algebra
Tuple Relational Calculus
{s|er_service_requests(s) ^ s.end_date = NULL}
Domain Relational Calculus

```
SELECT S.SERVICE_ID, S.START_DATE, S.END_DATE
FROM ER_SERVICE_REQUEST S
WHERE END_DATE IS NULL;
```

- Which employees have open service requests?
  Relational Algebra
  Tuple Relational Calculus

  Domain Relational Calculus

```
SELECT UNIQUE SA.EMPLOYEE_ID
FROM ER_SUBMITS_A SA INNER JOIN ER_SERVICE_REQUEST SR ON (SA.SERVICE_ID =
SR.SERVICE_ID)
WHERE SR.END_DATE IS NULL;
```

- What types of assets has service requests?

  Relational Algebra
  Tuple Relational Calculus
  Domain Relational Calculus

```
SELECT A.* FROM ER_A_TYPE A

WHERE EXISTS(SELECT UNIQUE IA.TYPE_ID

FROM ER_HAS_A HA INNER JOIN ER_IS_A IA ON(HA.ASSET_ID = IA.ASSET_ID));
```

Show Former Employees

Relational Algebra

Tuple Relational Calculus

Domain Relational Calculus

```
SELECT WF.EMPLOYEE_ID
FROM ER_WORKS_FOR WF INNER JOIN ER_DEPARTMENT D ON(WF.DEPARTMENT_ID =
D.DEPARTMENT_ID)
```

WHERE NOT WF.END_DATE IS NULL;

- Show Departments with more than 1 Employee

Relational Algebra
Tuple Relational Calculus
Domain Relational Calculus

```
SELECT D.*
FROM ER_DEPARTMENT D
WHERE EXISTS(SELECT EMPLOYEE_ID, COUNT(*) FROM ER_WORKS_FOR WF
WHERE WF.DEPARTMENT_ID = D.DEPARTMENT_ID
HAVING COUNT(*) > 1);
```

III. Implementation of Relational Database

a. SQL*PLUS

With the creation of the relational model completed, the description for each relation will be used to develop a database that will meet the requirements of the attributes, constraints, and relationships. We will use the Oracle Relational Database Management System implementation of SQL. Structured Query Language is a language used to manipulate data within a Relational Database Management System. SQL was first introduced by Edgar F. Codd in his 1970 paper, "A Relational Model of Data for Large Shared Data Banks." It was first developed at IBM in the 1970s. It has since gone through the standardization process. Today there are now many popular implementations including Microsoft's Transact-SQL, or T-SQL, MySQL, and Oracle. Oracle provides a command line interface called SQL*PLUS which allows SQL commands to be run interactively. This tool allows users to create and destroy a database very quickly.

b. Schema Objects in Oracle

The Oracle Database Management System uses a collection of schema objects to form a schema. A tablespace is used to logically organize the structure of the database. Schema objects are logical data structures which are stored in a tablespace within the database. The data for each schema object is stored in the tablespace's data files. This allows a tablespace to contain several different types of schema objects.

i. Tables

Tables represent relations from the relational model. They are the basic storage unit for an Oracle database. A table's columns represent the relation's attributes. The rows in the table represent a records or tuple in the relation. Each attribute has a unique name. The Tables store information about the relation's primary key, foreign keys, and constraints. After the table is created, data can be inserted into rows which represent the existence of tuples.

ii. Views

Views are virtual tables that return tuples from one or more tables. They are used when a command will be used repeatedly. Views do not use storage space like a table. Views can be used to display data that simplify the actual representation for users.

iii. Dimensions

Dimensions are relationships between columns in a table. This can be used between columns of that exist in the same table or in separate tables.

iv. Sequences

Sequence generators create a sequential set of numbers. The sequence numbers can then be used to determine order for queued operations or requests. They can be used to generate primary keys for a specific table.

v. Synonyms

Synonyms are aliases for different types of schema objects, such as tables, procedures, or views. They do not require any storage space. Synonyms can be used to hide internal data from outside users and can be used simplify SQL commands.

vi. Indexes

Databases optimize the traversal of each table by caching the values of unique attributes, such as primary keys. Indexes are used as additional attributes that will allow the database to more quickly accesses their values during comparisons. Indexes can also be created for combinations of certain attributes.

vii. Database Links

Database links are hard-coded, read-only links to other databases. These links allow one database to perform queries and retrieve results using another database.

viii. Stored procedures and functions

These can be used to automate repetitive tasks. A stored procedure or function always performs the same task as instructed upon its creation. Functions in Oracle always return a single value to the user, while stored procedures do not.

ix. Packages

Packages are a set of specific stored procedures, functions, and cursors. They act as a single unit of instructions. This is ideal for large-scale operations performed by stored procedures. Packages organize and simplify design requirements for databases that require complex tasks.

c. Schema Objects in this project

In this project, the two most frequently used schema objects are the table and the view. Most of the tables are created using syntax similar to this:

```
CREATE Table [TableName]
(
    attributes      attribute types         nullable? ,
    …               …                       … ,
    …               …                       … ,
Constraints:
    pk_tablename PRIMARY KEY (AttributeName)
    k_ParentName_ChildName FOREIGN KEY (AttributeName) REFERENCES ParentName
    (ParentAttributeName)
);
```

The scheme objects created are as follows:

ER_DEPARTMENT	Department Relation
ER_EMPLOYEE	Employee Relation
ER_SERVICE_REQUEST	Service Request Relation
ER_TECHNICIAN	Technician Relation
ER_SKILL	Skill Relation
ER_A_TYPE	Type Relation
ER_ASSET	Asset Relation
ER_WORKS_FOR	Works For Relation
ER_WORKS_ON	Works On Relation
ER_IS_A	Is A Relation
ER_HAS_A	Has A Relation
ER_SUBMITS_A	Submits A Relation
ER_POSSESSES	Possesses Relation

Following are the schemas and instances for each relation:

ER_DEPARTMENT

```
CS342 SQL> desc er_department;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 DEPARTMENT_ID                             NOT NULL NUMBER(4)
 DEPARTMENT_NAME                           NOT NULL VARCHAR2(25)
 LOCATION                                           VARCHAR2(30)

CS342 SQL> select * from er_department;

DEPARTMENT_ID DEPARTMENT_NAME        LOCATION
------------- ---------------------- ------------------------------------------------------
            1 ENGINEERING            CITY HALL
```

2	PLANNING	COMMUNITY DEVELOPMENT
3	WATER PRODUCTION	CORPORATION YARD
4	SEWER	WASTE WATER TREATMENT PLANT
5	STREETS	CORPORATION YARD

ER_EMPLOYEE

```
CS342 SQL> desc er_employee;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 EMPLOYEE_ID                               NOT NULL NUMBER(4)
 F_NAME                                    NOT NULL VARCHAR2(25)
 M_NAME                                             VARCHAR2(20)
 L_NAME                                    NOT NULL VARCHAR2(30)
 BIRTH_DATE                                NOT NULL DATE
 SOCIAL_SECURITY                           NOT NULL NUMBER(12)
 PHONE                                     NOT NULL NUMBER(10)
```

```
CS342 SQL> select * from er_employee;

EMPLOYEE_ID F_NAME               M_NAME          L_NAME                    BIRTH_DATE SOCIAL_SECURITY     PHONE
----------- ------------------------- --------------- ------------------------- --------- ---------------------------------- -------------------------
          1 Lucas                    Alexander       Skye                      01-JAN-70  123456789          6615550001
          2 Alvin                                    Chipmunk                  02-FEB-80  234567891         6615550002
          3 Simon                                    Chipmunk                  03-MAR-80  345678912          661555003
          4 Theodore                                 Chipmunk                  04-APR-80  456789123          6615550004
          5 Billy                    Joe             Armstrong                 05-MAY-71  567891234          6615550005
          6 Tim                                      Armstrong                 06-JUN-72  678912345          6615550006
          7 Scott                                    Weiland                   07-JUL-73  789123456          6615550007
          8 Tom                                      Morello                   08-AUG-74  891234567          6615550008
          9 Karen                    Lee             Orzolek                   22-NOV-78  912345678          6615550009
         10 Dave                                     Navarro                   09-SEP-74  129834765          6615550010
```

ER_SERVICE_REQUEST

```
CS342 SQL> DESC ER_SERVICE_REQUEST;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 SERVICE_ID                                NOT NULL NUMBER(4)
 START_DATE                                NOT NULL DATE
 END_DATE                                           DATE
 DESCRIPTION                                        VARCHAR2(30)
```

```
CS342 SQL> SELECT * FROM ER_SERVICE_REQUEST;

SERVICE_ID START_DATE END_DATE DESCRIPTION
---------- --------- --------- ------------------------------ --------- --------- ------------------------------
         1 01-JAN-10 02-JAN-10 WATER LEAK
         2 10-JAN-10 11-JAN-10 SEWER LEAK
         3 02-FEB-10 02-FEB-10 WATER LEAK
         4 12-FEB-10 13-FEB-10 SEWER LEAK
         5 03-MAR-10 03-MAR-10 MAINTENANCE
         6 13-MAR-10 14-MAR-10 MAINTENANCE
         7 04-APR-10 04-APR-10 MAINTENANCE
         8 14-APR-10 14-APR-10 WATER LEAK
         9 05-MAY-10 10-MAY-10 REBUILD
        10 15-MAY-10 16-MAY-10 WATER LEAK
        11 06-JUN-10 10-JUN-10 REBUILD
```

```
        12 07-JUL-10                       MAINTENANCE
        13 08-AUG-10                       MAINTENANCE
```

ER_TECHNICIAN

```
CS342 SQL> DESC ER_TECHNICIAN;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 EMPLOYEE_ID                               NUMBER(4)
 SKILL_ID                                  NUMBER(4)
```

```
CS342 SQL> SELECT * FROM ER_TECHNICIAN;

EMPLOYEE_ID     SKILL_ID
----------- ---------- ----------

          6          1
          7          2
          8          3
          9          4
          9          5
          1          2
```

ER_SKILL

```
CS342 SQL> DESC ER_SKILL;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 SKILL_ID                                  NOT NULL NUMBER(4)
 SKILL_NAME                                NOT NULL VARCHAR2(30)
 DESCRIPTION                                        VARCHAR2(30)
```

```
CS342 SQL> SELECT * FROM ER_SKILL;

  SKILL_ID SKILL_NAME                     DESCRIPTION
---------- ------------------------------ ------------------------------
         1 Water Operator                 State Certified
         2 Waste Water Operator           State Certified
         3 Maintenance Technician         Misc. Employee
         4 Data Entry                     Misc. Employee
         5 Street Technician              Misc. Employee
```

ER_A_TYPE

```
CS342 SQL> DESC ER_A_TYPE;
```

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------------------
TYPE_ID                                   NOT NULL NUMBER(4)
A_TYPE                                    NOT NULL VARCHAR2(25)
DESCRIPTION                                        VARCHAR2(30)
```

```
CS342 SQL> SELECT * FROM ER_A_TYPE;

   TYPE_ID A_TYPE                    DESCRIPTION
---------- ------------------------- ------------------------------
         1 Water Valve                Copper
         2 Water Line                PVC
         3 Sewer Valve                Copper
         4 Sewer Line                PVC
         5 Generator                 Gas Power
         6 Fire Hydrant               Cast Iron
```

ER_ASSET

```
CS342 SQL> DESC ER_ASSET;
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------------------
ASSET_ID                                  NOT NULL NUMBER(4)
INSTALL_DATE                              NOT NULL DATE
LAST_MAINTENANCE                                   DATE
LOCATION                                           VARCHAR2(25)
```

```
CS342 SQL> SELECT * FROM ER_ASSET;

  ASSET_ID INSTALL_DA LAST_MAINT LOCATION
---------- --------- --------- -------------------------
         1 01-JAN-81 12-DEC-09 CORPORATION YARD
         2  02-FEB-82 11-NOV-09 LEXINGTON  AVE
         3 03-MAR-99 10-OCT-08 JEFFERSON ST
         4  04-APR-04 04-MAY-10 DCCF
         6 02-FEB-02 03-MAR-03 CECIL AVE  ST
         7 10-OCT-90 12-DEC-02 CITY HALL
         8 11-NOV-01
         9 06-JUN-96 02-FEB-07
        10 09-SEP-99 09-SEP-09
         5 05-MAY-05 01-JAN-09 WWTP
```

ER_WORKS_FOR

```
CS342 SQL> DESC ER_WORKS_FOR;
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------------------
EMPLOYEE_ID                               NOT NULL NUMBER(4)
DEPARTMENT_ID                             NOT NULL NUMBER(4)
START_DATE                                NOT NULL DATE
END_DATE                                           DATE
```

```
CS342 SQL> SELECT * FROM ER_WORKS_FOR;
```

EMPLOYEE_ID	DEPARTMENT_ID	START_DATE	END_DATE
1	5	01-JAN-01	11-NOV-09
2	4	02-FEB-02	
5	3	03-MAR-03	
4	1	04-APR-04	
3	2	05-MAY-05	
7	4	06-JUN-06	
6	3	07-JUL-07	
8	1	08-AUG-08	
9	2	09-SEP-09	
10	5	10-OCT-10	

ER_WORKS_ON

CS342 SQL> DESC ER_WORKS_ON;

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(4)
SERVICE_ID	NOT NULL	NUMBER(4)
START_DATE	NOT NULL	DATE
END_DATE		DATE

CS342 SQL> SELECT * FROM ER_WORKS_ON;

EMPLOYEE_ID	SERVICE_ID	START_DATE	END_DATE
6	1	02-JAN-10	02-JAN-10
6	7	04-APR-10	04-APR-10
6	10	16-MAY-10	16-MAY-10
6	3	02-FEB-10	02-FEB-10
6	13	10-AUG-10	
7	2	11-JAN-10	11-JAN-10
7	4	13-FEB-10	13-FEB-10
7	8	14-APR-10	14-APR-10
7	11	08-JUN-10	10-JUN-10
7	12	10-JUL-10	
8	5	03-MAR-10	03-MAR-10
6	6	13-MAR-10	13-MAR-10
6	9	07-MAY-10	09-MAY-10

ER_IS_A

CS342 SQL> DESC ER_IS_A;

Name	Null?	Type
ASSET_ID	NOT NULL	NUMBER(4)
TYPE_ID	NOT NULL	NUMBER(4)

CS342 SQL> SELECT * FROM ER_IS_A;

ASSET_ID	TYPE_ID
1	5
2	2
3	4
4	3
5	3
6	1
7	5

```
         8          1
         9          6
        10          6
```

ER_HAS_A

```
CS342 SQL> DESC ER_HAS_A
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 SERVICE_ID                                NOT NULL NUMBER(4)
 ASSET_ID                                  NOT NULL NUMBER(4)
```

```
CS342 SQL> SELECT * FROM ER_HAS_A;

SERVICE_ID   ASSET_ID
---------- ----------
         1          8
         2          4
         3          2
         4          3
         5          7
         6          9
         7          6
         8          4
         9         10
        10          8
        11          5
        12          3
        13          2
```

ER_SUBMITS_A

```
CS342 SQL> DESC ER_SUBMITS_A
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 EMPLOYEE_ID                               NOT NULL NUMBER(4)
 SERVICE_ID                                NOT NULL NUMBER(4)
```

```
CS342 SQL> SELECT * FROM ER_SUBMITS_A;

EMPLOYEE_ID     SERVICE_ID
-----------     ----------
          1              1
          2              2
          3              3
          4              4
          1              5
          2              6
          3              7
```

```
4                    8
1                    9
2                    10
3                    11
4                    12
5                    13
```

ER_POSSESSES

```
CS342 SQL> DESC ER_POSSESSES;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------
 EMPLOYEE_ID                               NOT NULL NUMBER(4)
 SKILL_ID                                  NOT NULL NUMBER(4)
 START_DATE                                NOT NULL DATE
 END_DATE                                           DATE
```

```
CS342 SQL> SELECT * FROM ER_POSSESSES;

EMPLOYEE_ID  SKILL_ID  START_DAT E  END_DATE
-----------  ----------  ---------  ---------
      6          1        08-AUG-08
      7          2        07-JUL-07
      8          3         08-AUG-08
      9          4        09-SEP-09
      9          5        10-OCT-10
      1          2        02-FEB-02          10-OCT-09
```

d. SQL Queries

Select all technicians who worked on service requests.

```
SELECT T.EMPLOYEE_ID
FROM ER_TECHNICIAN T
WHERE EXISTS(SELECT * FROM ER_WORKS_ON W WHERE W.EMPLOYEE_ID = T.EMPLOYEE_ID);
```

```
EMPLOYEE_ID
-----------
      6
      7
      8
```

Select all technicians who have more than 2 service requests.

```
SELECT P.*
FROM ER_POSSESSES P
WHERE EXISTS(SELECT EMPLOYEE_ID, COUNT(*)
      FROM ER_WORKS_ON W
      WHERE P.EMPLOYEE_ID = W.EMPLOYEE_ID
      HAVING COUNT(*) > 1);
```

EMPLOYEE_ID	SKILL_ID	START_DATE	END_DATE
6	1	08-AUG-08	
7	2	07-JUL-07	

Select All employees who have submitted a service request.

```
SELECT E.EMPLOYEE_ID
FROM ER_EMPLOYEE E
WHERE EXISTS (SELECT * FROM ER_SUBMITS_A S
WHERE S.EMPLOYEE_ID = E.EMPLOYEE_ID);
```

EMPLOYEE_ID
1
2
3
4
5

Show what skills all technicians have

```
SELECT P.SKILL_ID, P.EMPLOYEE_ID
FROM ER_POSSESSES P;
```

SKILL_ID	EMPLOYEE_ID
1	6
2	7
3	8
4	9
5	9
2	1

Show Employees with no skills

```
SELECT E.F_NAME, E.L_NAME
FROM ER_EMPLOYEE E
WHERE NOT EXISTS(SELECT * FROM ER_POSSESSES P WHERE P.EMPLOYEE_ID = E.EMPLOYEE_ID);
```

F_NAME	L_NAME
Billy	Armstrong
Dave	Navarro
Theodore	Chipmunk
Alvin	Chipmunk
Simon	Chipmunk

Show all open service requests

```
SELECT S.SERVICE_ID, S.START_DATE, S.END_DATE
```

```
FROM ER_SERVICE_REQUEST S
WHERE END_DATE IS NULL;
```

SERVICE_ID	START_DATE	END_DATE
12	07-JUL-10	
13	08-AUG-10	

Which employees have open service requests?

```
SELECT UNIQUE SA.EMPLOYEE_ID
FROM ER_SUBMITS_A SA INNER JOIN ER_SERVICE_REQUEST SR ON (SA.SERVICE_ID = SR.SERVICE_ID)
WHERE SR.END_DATE IS NULL;
```

EMPLOYEE_ID
5
4

What types of assets has service requests?

```
SELECT A.* FROM ER_A_TYPE A
WHERE EXISTS(SELECT UNIQUE IA.TYPE_ID
FROM ER_HAS_A HA INNER JOIN ER_IS_A IA ON(HA.ASSET_ID = IA.ASSET_ID));
```

TYPE_ID	A_TYPE	DESCRIPTION
1	Water Valve	Copper
2	Water Line	PVC
3	Sewer Valve	Copper
4	Sewer Line	PVC
5	Generator	Gas Power
6	Fire Hydrant	Cast Iron

Show Former Employees

```
SELECT WF.EMPLOYEE_ID
FROM ER_WORKS_FOR WF INNER JOIN ER_DEPARTMENT D ON(WF.DEPARTMENT_ID =
D.DEPARTMENT_ID)
WHERE NOT WF.END_DATE IS NULL;
```

EMPLOYEE_ID
1

Show Departments with more than 1 Employee

```
SELECT D.*
FROM ER_DEPARTMENT D
WHERE EXISTS(SELECT EMPLOYEE_ID, COUNT(*) FROM ER_WORKS_FOR WF
WHERE WF.DEPARTMENT_ID = D.DEPARTMENT_ID
HAVING COUNT(*) > 1);
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION
1	ENGINEERING	CITY HALL
2	PLANNING	COMMUNITY DEVELOPMENT

3	WATER PRODUCTION	CORPORATION YARD
4	SEWER	WASTE WATER TREATMENT PLANT
5	STREETS	CORPORATION YARD

IV. Stored Subprograms, Packages and Triggers

a. Common Features in Oracle PL/SQL and Microsoft Transact-SQL

Oracle PL/SQL and Microsoft's Transaction has evolved overtime to complement the needs of developers. Through the use of standardization the two languages have a comparable set of tools for use by developers. Both languages support basic commands to create tables, constraints, and functions. The also support commands such as cursors, stored procedures, triggers, and packages. The primary differences are in the syntax used to create and maintain these objects in the database.

Stored subprograms, or stored procedures, are supported in both PL/SQL and T-SQL and are used for tasks that can be run repeatedly and quickly by specific users. Tasks can typically consist of inserting, deleting, or updating records in the database. By only allowing the users to interface with the subprograms sensitive information can be protected.

b. Oracle PL/SQL

Most PL/SQL sub programs follow a similar syntax for declaration. Code statements are organized into blocks and there are three main sections of a block:

- Declaration: Declaration of variables, cursors, and user-defined exceptions are made here.
- Execution: This portion consists of the SQL statements that perform the task's job.
- Exception: This section catches any exceptions, either system or user-defined, raised during execution of the task.

Layout:

```
DECLARE
        variable_name                   variable_type                   := value | DEFAULT
BEGIN
        SELECT | INSERT | UPDATE | DELETE
END;
```

Variable types:

Oracle PL/SQL support standard types such as numbers, floating points, character arrays, dates, and unique IDs.

Cursors:

They are user-defined SQL statements that allow traversal of a table using a loop structure. They are defined with the following syntax.

```
DECLARE
        CURSOR cursor_name [parameters]
        IS select_statement;
After creation, a cursor can be used in the following format:
BEGIN
        FOR t in cursor_name LOOP
                Perform tasks
        END LOOP;
END;
```

Control statements

These are used to manage the logic of a PL/SQL subprogram. With all procedural languages the location of each statement can cause unexpected results. The following are example control statements:

```
IF condition THEN statement;
ELSEIF condition THEN statement;
END IF;
LOOP
        EXIT WHEN can be used to quit this loop
END LOOP;
FOR I IN lowerbound .. upperbound LOOP
        statement
END LOOP;
FOR cursor_variable IN cursor_name LOOP
        statement
END LOOP;
Exception Handling
```

In PL/SQL users can also catch and raise exceptions. The syntax to raise and handle exceptions is as follows:

```
DECLARE
        User_defined_exception EXCEPTION;
BEGIN
        IF condition THEN RAISE User_defined_exception;
        END IF;
EXCEPTION
        WHEN Exception_name THEN statement;
END;
```

Stored procedures

Each Stored procedures can perform complex tasks on the database while maintaining abstraction. The structure of a stored procedure depends on the type of work it will be performing.

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ (variablename                IN|OUT                variabletype)]
AS
(DECLARE variables go here)
BEGIN
        SQL statements
```

```
END;
```

Stored functions

Stored functions are very similar to stored functions with the exception that they return a value. The following is the syntax for a stored function.

```
CREATE [OR REPLACE] FUNCTION function_name
[ (variablename          IN|OUT          variabletype)]
RETURN datatype;
AS
(DECLARE variables go here)
BEGIN
        SQL statements;
        RETURN variable;
END;
```

Packages

Packages are a collection of stored procedures and stored functions.

```
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 allow for the easy collection of records, logs, and audits. Triggers are executed when a specific condition is met. These typically include UPDATE, DELETE, and INSERT. Once the triggers are created they will automatically perform. There is no need to maintain or check data before or after the operations.

```
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;
```

c. Oracle PL/SQL Subprograms

I have create several subprograms for this database project. Included are 2 Stored Procedures, 4 Functions, and 1 trigger.

Stored Procedures

ER_SP_INSERT_EMPLOYEE

```
CREATE OR REPLACE PROCEDURE er_sp_insert_employee
(
        empID                   IN NUMBER,
        fName       I           N VARCHAR2,
        DOB                     IN DATE,
        oPhone                  IN VARCHAR2,
        cPhone                  IN VARCHAR2,
         eMAIL                  IN VARCHAR2,
         deptID                 IN NUMBER,
         sDate                  IN DATE,
         eDATE                  IN DATE
)
AS
BEGIN
 INSERT INTO er_employee
 VALUES
 (
                        empID,
                        fName,
                        DOB,
                        oPhone,
                        cPhone,
                        eMAIL
 );
 INSERT INTO er_works_for
 VALUES
 (
        empID,
        deptID,
        sDate,
        eDate
 );
 EXCEPTION
        WHEN OTHERS THEN
        ROLLBACK;
          raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
                        '-ERROR-' || SQLERRM );
END er_sp_insert_employee;
/
```

ER_SP_INSERT_SERVICE_REQUEST

```
CREATE OR REPLACE PROCEDURE er_sp_insert_service_request
(
 servID             IN NUMBER,
 empID              IN NUMBER,
 assetID            IN NUMBER,
 sDate              IN DATE,
 eDate              IN DATE,
```

```
 descn              IN VARCHAR2
)
AS
BEGIN
 INSERT INTO er_service_request
 VALUES
 (
                    servID,
                    sDate,
                    eDate,
                    descn
 );
 INSERT INTO er_submits_a
 (
        empID,
        servID

 );
 INSERT INTO er_has_a
 (
        servID,
        assetID

 );
 EXCEPTION
        WHEN OTHERS THEN
        ROLLBACK;
          raise_application_error( -40001, 'An error occurred in ' || SQLCODE ||
                    '-ERROR-' || SQLERRM );
END er_sp_insert_service_request;
/
```

FUNCTIONS

ER_FUN_GETSKILLS

This function returns the number of skills an employee has.

```
create or replace function er_fun_getskills(empID IN number)
 return number
 is numSkills number(10);
 begin
   select count(*)
   into numskills
   from er_technician
   where er_technician.employee_id = empID;
   return(numSkills);
 end;
/
```

ER_FUN_GETEMPLOYEES

This function returns the number of employees a department has.

```
create or replace function er_fun_getemployees(deptID IN number)
 return number
 is numEmployees number(10);
 begin
   select count(*)
   into numEmployees
   from er_works_for
```

```
   where er_works_for.department_id = deptID;
   return(numEmployees);
 end;
/
```

ER_FUN_GETASSETS

This function returns the number of assets of a specific type

```
create or replace function er_fun_getassets(typeID IN number)
 return number
 is numAssets number(10);
 begin
   select count(*)
   into numAssets
   from er_is_a
   where er_is_a.type_id = typeID;
   return(numAssets);
 end;
/
```

ER_FUN_GETSKILLAVG

This function returns the average skills each technician has in a department

```
CREATE OR REPLACE FUNCTION er_fun_getskillavg(deptID IN NUMBER)
RETURN NUMBER
IS
avgskills NUMBER(10) := 0;
no_skills NUMBER(10) := 0;
no_employees NUMBER(10) := 0;
CURSOR employees IS
 SELECT employee_ID, department_ID FROM er_works_for ORDER BY department_id;
BEGIN
avgskills := 0;
no_skills := 0;
no_employees := er_fun_getemployees(deptID);
FOR e IN employees LOOP
 IF e.department_id = deptID  THEN no_skills := no_skills + er_fun_getskills(e.employee_ID);
 END IF;
END LOOP;
avgskills := trunc(no_skills/no_employees, 3);
RETURN(avgskills);
END;
/
```

TRIGGERS

This trigger tracks updates made to ER_SERVICE_REQUEST

ER_TR_UPDATE_SERVICE_REQUEST

```
CREATE OR REPLACE TRIGGER er_tr_update_service_request
```

```
BEFORE UPDATE
 ON er_service_request
 FOR EACH ROW
BEGIN
 INSERT INTO er_sr_logtable
 VALUES(:old.service_id, :old.end_date);
END;
/
```

V. Graphical User Interface Design and Implementation

a. Daily Activities

This application will service many positions within the organization.

i. Administrative Staff

Administrative Users will be able to submit service requests for and generate reports. They will be able to view Assets by Type, View Employees, and Submitted Service Requests.

ii. Technicians

Technicians can add/update the service requests, view all service requests, and view their certifications.

iii. Managers

The Management Team can add/delete all items in the database. These items include employees, assets, and service requests. They can keep track of their technicians certifications.

b. Relations, Views and Subprograms

In order to meet the needs and requirements this application will access most of the relations that are built into the database. Each of these relations will have a corresponding Table Adapter to facilitate data access and manipulation.

i. RELATIONS

ER_A_TYPE
ER_ASSET
ER_DEPARTMENT
ER_HAS_A
ER_IS_A
ER_SERVICE_REQUEST
ER_SKILL
ER_SUBMITS_A
ER_TECHNICIAN
ER_WORKS_FOR
ER_WORKS_ON

ii. VIEWS

ER_VW_AST_TYP
ER_VW_EMP_DEPT
ER_VW_SR_AST_TYP
ER_VW_TEC_SKL_DEPT
ER_VW_UP_EMP_DEPT

iii. SUBPROGRAMS

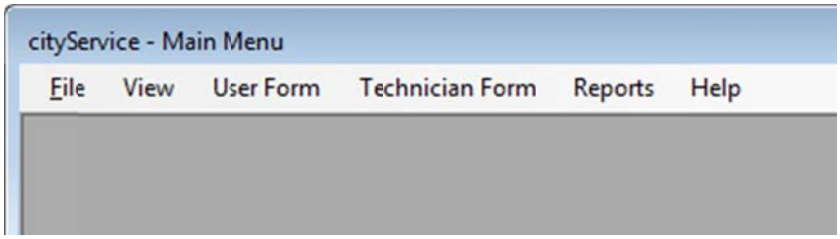
ER_SP_DELETE_TECHNICIAN
ER_SP_INSERT_ASSET
ER_SP_INSERT_EMPLOYEE
ER_SP_INSERT_SERVICE_RECORD
ER_SP_UPDATE_EMPLOYEE
ER_SP_UPDATE_SERVICE_RECORD

These relations are used in the application to display the data in a meaningful way for the users. The Views are used to combine relations together to represent information that is used to perform tasks

within the application. The tasks include viewing which skills a technicians possess by department. The service requests submitted along with the assets and type.

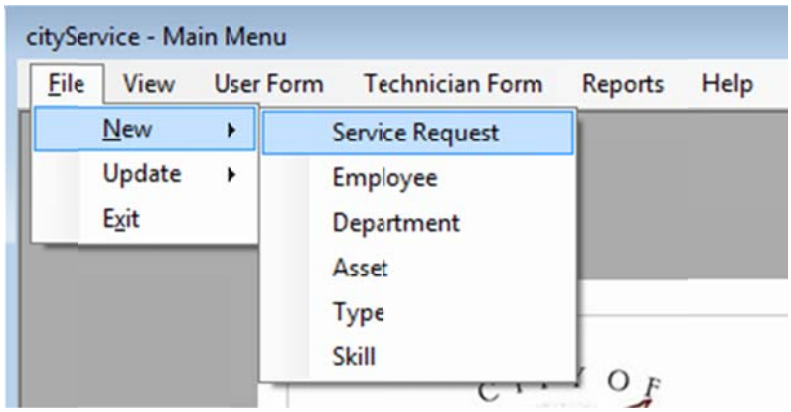
c. Screen Shots

The applications interface is organized using a menu strip. The menu consists of options for File, View, User, Technician, Report and Help. An overview will be given of each of the menu options. The Main Form is a Multiple Document Interface. Each of the Windows forms that are selected via the menu options are child forms.



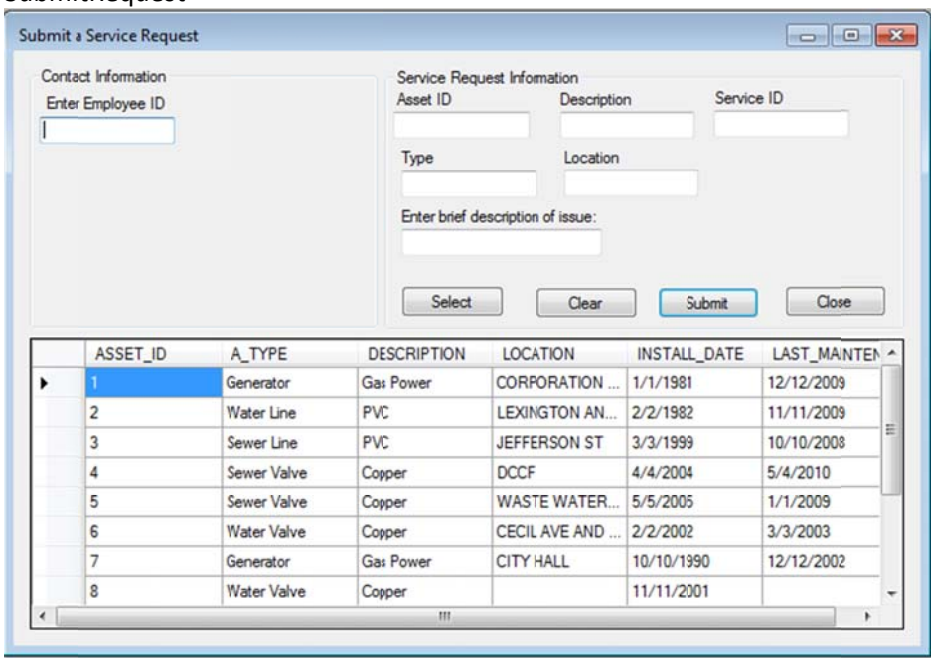
File -> New

These options can be used to create new objects within the system. Several stored procedures are used to insert data into the database. The stored procedures are a more efficient method to use for repetitive tasks.



Each of the options has a Windows Form to perform the tasks offered. The most complex forms include SubmitRequest and Add Employee.

SubmitRequest

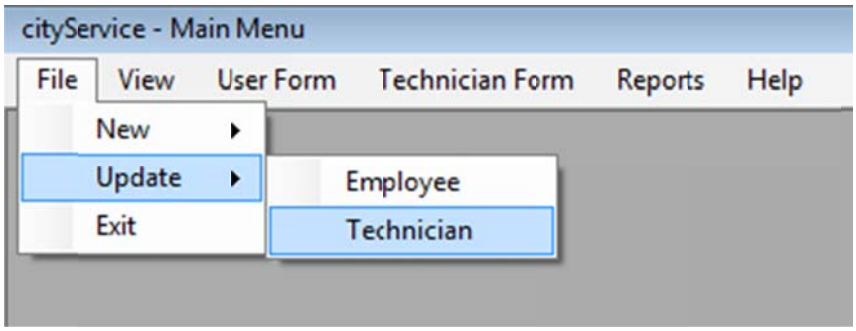


This form is used to add a new Service Request. A user will need to enter their employee ID. They will browse for the asset. This Data Grid View uses a view, ER_VW_AST_TYP, in order to display they required information. A user can enter their employee ID, browse for an asset, and provide a brief description of the issue. If they make a mistake prior to submitting their request, then they can clear all entered information to start the process over.

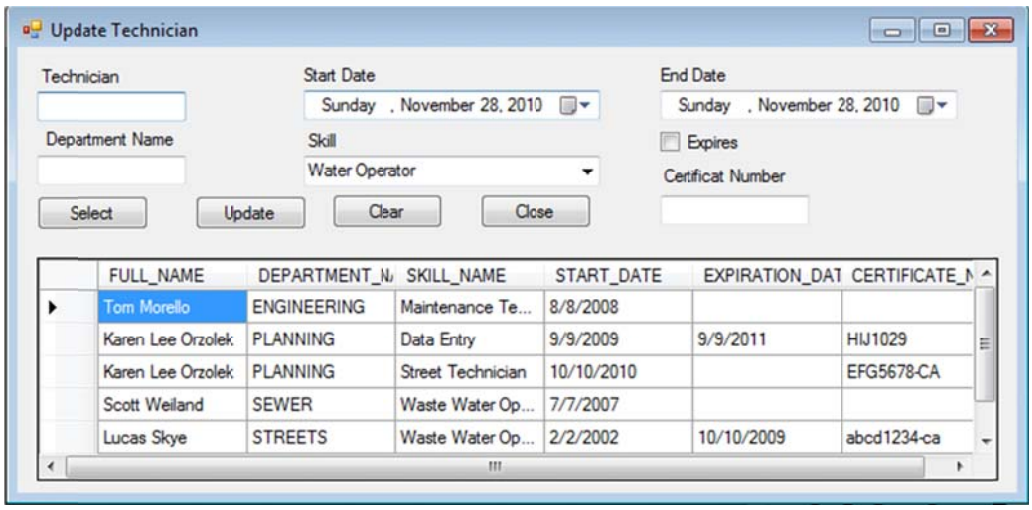
Once an asset is selected its information will be populated within the Service Request Information Group Box.

File -> Update

To update an Employee or a Technician you can navigate to these forms. These options also used stored procedures to update the necessary tables within the database.

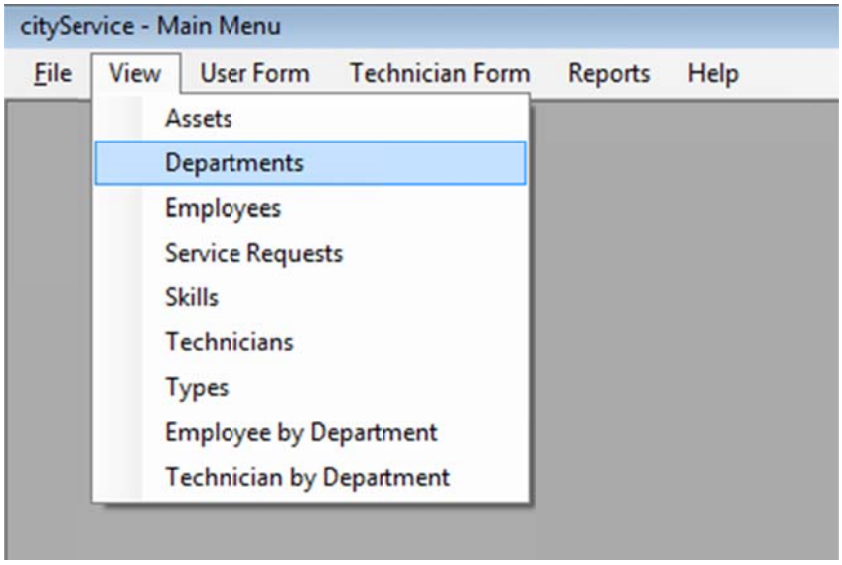


UpdateTechnician

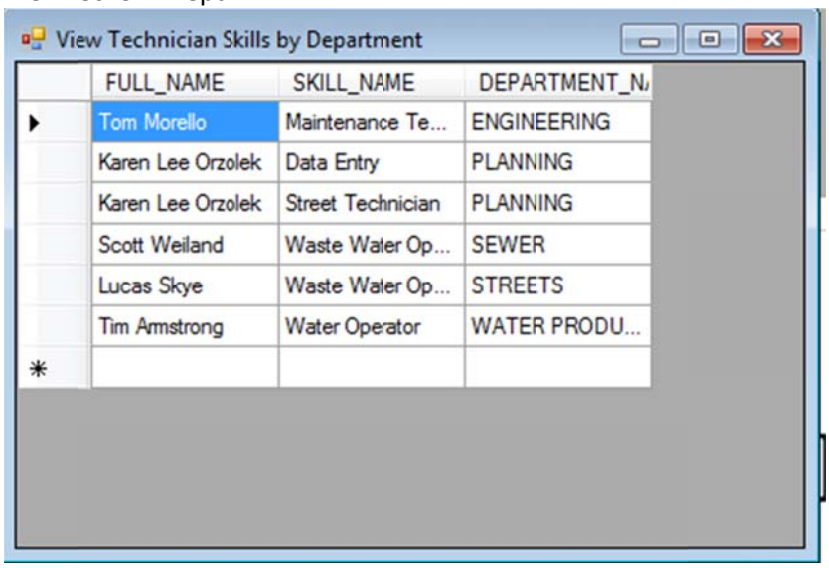


This form can be used to add a skill to an existing Technician as well as modify any attribute associated with their skills, including expiration dates and certification numbers. This form uses view ER_TEC_SKL_DEPT in the Data Grid View to display the information provided. Here you can select the technician that will be update. Once selected, a technicians information will populate some of the text fields, then a new skill can be added by selecting the appropriate skills from the combo boxes.

The application also has forms to view data within the database. These forms use Data Grid Views to either display data directly in the tables or use views to construct the necessary arrangement of data.

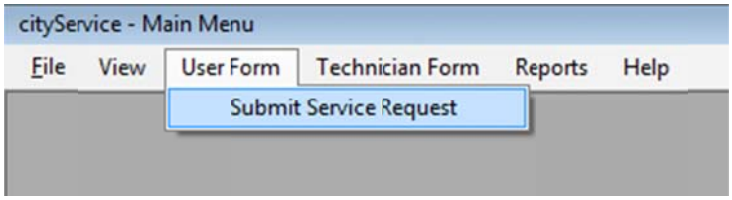


ViewTechSkillDept



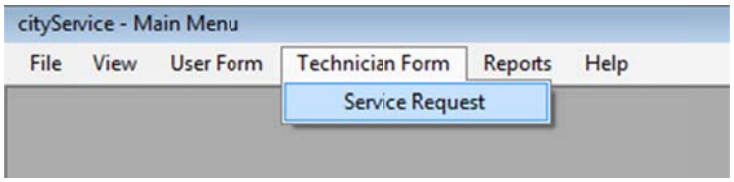
This form uses a view ER_VW_TEC_SKL_DEPT to display technicians along with their skills and the departments they work for.

User Form -> Submit Service Request



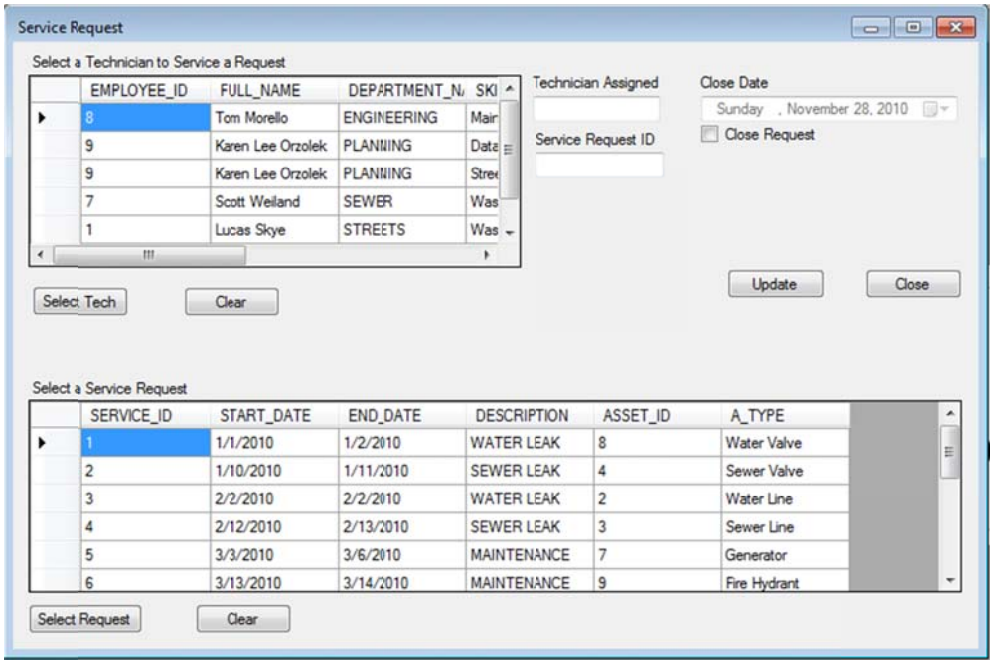
This option uses the same windows form as File -> New -> Service Request

Technician Form -> Service Request



The Technician Forms menu is used to allow Technicians to access the information needed for them to do their job.

ServiceRequest



This form will be used to assign a technician to a service request as well as closing out the request. The form uses two different Data Grid Views in conjunction with database views to display meaningful information in order to facilitate the needed actions.

d. Description of Code

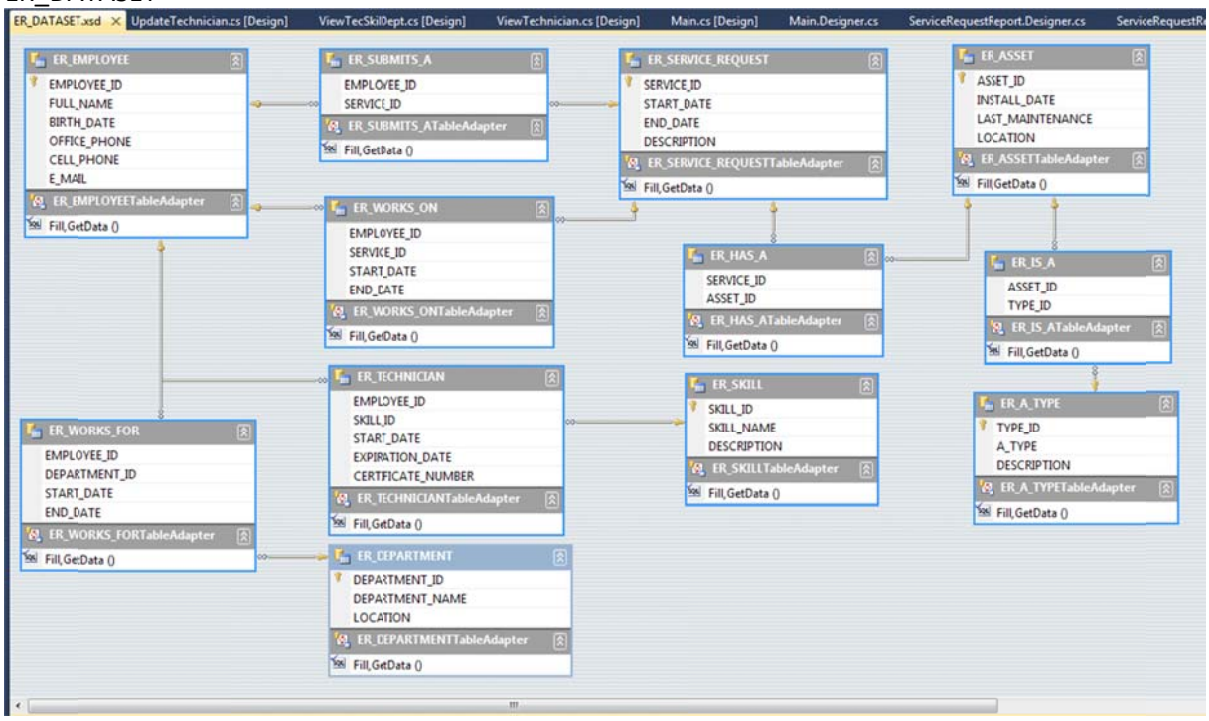
The design of the graphical user interface was started early in the development process. This was done to ensure the application had an intuitive look and feel to it. The application uses a Multiple Documents Interface which uses a parent form and several child forms. To navigate within the application a series of menu items are used on a tool strip. Each option on the tool strip is meant to represent a series of functions. These include creating new objects, viewing data, and working with the data.

The application was written using in C# under Visual Studio 2010. This is Microsoft's latest release of its popular Integrated Development Environment. In order to connect to the Oracle Database on Helios, I installed Oracle's development tools for Visual Studio, Oracle Data Access Components with Oracle Developer Tools for Visual Studio 11.2.0.1.2. These tools allowed a seamless integration of Oracle data components into the Visual Studio IDE.

In order to connect to the database a connection string is used. The connection string consists of many parameters such as the hostname, port number, and username/password. The connection string allows the application to gain access to the appropriate resources on the host.

A dataset was created to access the data components with in the database.

ER_DATASET



The dataset contains table adapters for each of the tables that are accessed by the application. Table adapters provide needed functionality such as filling the data grid views.

The application consists of many different forms that accomplish specific tasks. These forms include:

AddEmployee
AddDepartment
AddAsset
AddType
AddSkill
Main
ServiceRequest
ServiceRequestReport
SubmitRequest
Splash
UpdateEmployee
UpdateTechnician
ViewAssets
ViewDepartments
ViewEmpDept
ViewEmployees
ViewServiceRequests
ViewSkills
ViewTechnician
ViewTechSkillDept
ViewTypes
AboutUs

I tried to use friendly names and describe the purpose of each form as it was developed. A general overview will be given of the main forms.

Main
This is the parent MDI form. From it you can gain access to the menu tool strip and all child forms.

SubmitRequest
This child form is used to submit a service request. It is a friendly interface that users can navigate. It consists of several command buttons that have event handlers that are used to make appropriate calls to the database.

AddEmployee
This child form is used to add an employee to the database. It provides a friendly interface is used to provide a one stop form for adding an employee. Combo drop box is used to house the departments which an employee can be a part of. Most of the forms that are used to "Add" objects have the same look and feel. This consistency is ideal for staff to learn how to operate the application.

ServiceRequest
This child form is used by technician and managers. It allows a technician to be assigned to a service request. It also provides all service requests in a data grid view. A technician can browse the submitted requests and view all data pertaining to the request.

UpdateEmployee
This child form is used to modify any attribute of an employee. These attributes include phone numbers, e-mail address, and which department they work for. I tried to use the same general layout of the Add forms for consistency and ease of use.
ViewTechSkillDept
This child form provides information on the technicians. It uses a data grid view to display the technician along with their skills and which department they work for. Just as the Add forms all

provide similar functionality. I wanted all the “View” forms to display the data in a meaningful way. I want to provide a friendly interface that allows users to gain access to the information they need.

e. Major Steps in Design and Implementation

I set out to develop a friendly interface to a robust database. The features incorporated into the application are needed in the day to day operations of the department for which it was developed. I chose to use an MDI application because I did not want to clutter the desktop of the users. I wanted to provide an application that was intuitive and does not require a lot of technical knowhow. The users of the application will be focused on performing specific tasks so I wanted to ensure that they are provided all the functionality to complete each process within one interface.

Microsoft’s Visual Studio 2010 is a robust IDE with many features built into it. The IDE auto generates code and has a feature called intelli-sense. This allows you to quickly find the appropriate handlers and methods.

I started developing this application first by creating a basic structure for how the interface was going to be presented. I initially did not concern myself with how it would function. Staying focused with the layout and the look and feel of the application.

Once I was comfortable with the basic a layout of the application I connected it to the database using the connection string. The first forms I created were the View forms. I wanted to ensure that the application was communicating with the database in an appropriate manner.

After verifying that the data was being represented correctly I began working on AddEmployee form. I used this form as template for the way the rest of the forms were going to be laid out. This is done to ensure that all forms maintained a consistency the users would be comfortable with. This process continued with each of the Add forms and with the Update Forms.

The SubmitRequest and ServiceRequest forms were created to provide two different aspects of a service request. The SubmitRequest is used to simplify submitting a service request. I did not want users to have to navigate several form to complete this task. The same thought went behind the ServiceRequest form.

Conclusion

This project was conceived out of a need to automate the day to day operations in the Public Works Department at the City of Delano. I had been approached by the Department Head to assist them and develop an application that will allow their staff to keep track of their daily activities. This project has served as a very powerful mile stone. I will now begin to migrate this application to Microsoft SQL Server and integrate it with ESRI ArcServer. It is my hope that I can provide a rich internet application that allows its users to work smarter.