INSTRUCTION SHEET

To connect to the database server, use the following command at the command prompt.

telnet cis400b.cis.ysu.edu

To enter the interactive SQL mode type:

strsql

To create a table, use the following syntax:

CREATE TABLE table_name
(
    attribute datatype,
    attribute datatype,
    attribute datatype,
    ...
    PRIMARY KEY(column)
)

Example:

CREATE TABLE Department
(
    dept_id INT,
    dept_name VARCHAR(20),
    dept_building VARCHAR(20),
    PRIMARY KEY(dept_id)
)

Now we have a table structure, but the table isn’t holding any information yet. To put values into the table, use the following command:

INSERT INTO table_name VALUES ('value', 'value', 'value', ....)

Example:

INSERT INTO Department VALUES ('10', 'CSIS', 'Meshel Hall')

Go ahead and enter a few sets of values into the table. Note that for the dept_id field you cannot use the same value twice. This is because this field has been designated as the primary key. The primary key field cannot contain duplicates.

If you need to remove a record that you have created, you can use the following command:
DELETE FROM table_name WHERE attribute=('value')

EXAMPLE:

DELETE FROM Department WHERE dept_id=('10')

Now let’s have the database system show us the values we entered into this table. For this we will use a SELECT statement.

SELECT * FROM table_name

Example: SELECT * FROM Department

The * stands for all. You are telling the system that you want to see the values for all of the attributes for each record in this table. To limit that attributes that are displayed you can be more specific.

SELECT attribute, attribute, … FROM table_name

SELECT dept_name, dept_building FROM Department

Perhaps you want to limit the records that are displayed rather than the attributes. If you want to only view the records with a certain value, you would use the following kind of SELECT statement.

SELECT * FROM table_name WHERE attribute=('value')

Example:

SELECT * FROM Department WHERE dept_id=('10')

You can combine both of these ideas and create a query like the following:

SELECT dept_name, dept_building FROM Department WHERE dept_id=('10')

Suppose you need to change values that have been assigned within a record but you don’t want to delete and recreate the record with the new values. To accomplish this you can use an UPDATE statement with the following syntax:

UPDATE table_name SET attribute=('value') WHERE attribute=('value')

Example:

UPDATE Employee SET first_name=('Tony') WHERE emp_id=('5900')
Now let’s create a second table, called **Employee** that has attributes such as **emp_id, first_name, last_name** and **dept_id**. The code for this table should look similar to the following:

```
CREATE TABLE Employee
(
    emp_id INT,
    first_name VARCHAR(20),
    last_name VARCHAR(20),
    dept_id INT,
    PRIMARY KEY(emp_id),
    FOREIGN KEY(dept_id) REFERENCES Department (dept_id)
)
```

Note that this table has a foreign key. A foreign key is typically a primary key from another table. It is useful when forming queries across multiple tables. This introduces a concept known as referential integrity. It prevents a user from inserting a record with a value in the foreign key field that doesn’t already exist in the primary key field of the referenced table. In this case, it prevents a user from entering a record into the Employee table with a dept_id that does not already exist in the Department table. The iSeries enforces referential integrity by default.

Insert some records into the newly created Employee table. One possible **INSERT** statement would look like:

```
INSERT INTO Employee VALUES ('5900','Troy', 'Smith', '10')
```

Once we have a few values in each table, we can begin to form more complex queries. This is where we can really start to see the power of a relational database. An example query could be:

```
SELECT E.first_name, E.last_name, D.dept_name FROM Employee E, Department D
WHERE D.dept_id=('10') AND E.dept_id=('10')
```

As you can see, we have pulled related values from two tables in a way that assigns them some meaning. This is just a small example of what you can do with a relational database system.