ASSIGNMENT-1

NOTE:

1. GREY COLOR LINES - COMMENTS FOR THE QUERY TYPE QUESTIONS.
2. BLUE COLOR LINES - QUERIES.
3. RED COLOR LINES - SCHEMA

1. Relational Model

1.1 Representing the Schema for each Relation in the University Database

Schema for the Student Relation

Student(sno, Surname, givename, major)

Schema for the Staff Relation

Staff(eno, surname, givename, department, rank)

Schema for the Class Relation

Class(cno, lecturer*, day, time, room)

Schema for the Enrol Relation

Enrol(sno*, cno*, grade)

1.2 Creating the Tables for each relation using the CREATE statement:

Before creating a table it is mandatory to get assured that there is no duplicate table in the database (i.e., pre created table with the same table name that we are going to create). In order to achieve this we simply drop the tables in the order of their dependencies (i.e., taking constraints like foreign keys into consideration) by using the Drop statement.

Syntax:- drop table tablename;

```
--Therefore,

--As Enrol table has the key constraints as foreign keys
--from the Class table and Student table, it is in the
highest --level and it should be dropped first.

drop table Enrol;

--Similar, in the case of Class table which has the keys
--from Staff table.

drop table Class;

--Dropping the remaining tables as usual as they are
--independent tables.

drop table Staff;

drop table Student;
```

```
-- Now Creating the required tables in the order of
--Student, Staff, Class and Enrol, this is because of
--their dependencies which requires one table to
--create before creating another
-- Create table for the Student Relation
create table Student (
sno varchar(6) primary key,
surname char(12),
givename char(10),
major char(10)
);

--Create table for the Staff Relation
create table Staff(
eno varchar(6) primary key,
surname char(20),
givename char(20),
department char(30),
rank char(30)
);
```

```
--Create table for the Class Relation
create table Class(
cno varchar(20) primary key,
lecturer varchar(10),
day varchar(10),
time varchar(10),
room varchar(15),
CONSTRAINT Staff_eno_fk foreign key(lecturer) references Staff(eno)
);
--Create table for Enrol Relation
create table Enrol (
sno varchar(10),
cno varchar(10),
grade varchar(10) UNIQUE,
CONSTRAINT student_sno_fk foreign key(sno) references Student(sno),
CONSTRAINT class_cno_fk foreign key(cno) references Class(cno),
CONSTRAINT table_pk primary key(sno, cno)
);
```

1.3 INSERT INTO Statements to create a sample database

```
--Insert data into Student Table

insert into Student(sno, surname, givename, major)
values('s1001', 'Smith', 'Tom', 'History');

insert into Student(sno, surname, givename, major)
values('s1002', 'Chin', 'Ann', 'Maths');

insert into Student(sno, surname, givename, major)
values('s1003', 'Lee', 'Perry', 'Arts');

insert into Student(sno, surname, givename, major)
values('s1005', 'Smith', 'John', 'History');

insert into Student(sno, surname, givename, major)
values('s1006','River','Jane','Arts');

--Insert Data into Staff Table

insert into Staff(eno, surname, givename, department,
rank)
values('e123','Bowl','Alex','Maths','Lecturer');

insert into Staff(eno, surname, givename, department,
rank)
values('e205','Cox','Kevin','CSC','Associate Professor');

insert into Staff(eno, surname, givename, department,
rank)
values('e301','Jones','David','Arts','Senior Lecturer');
```

```
--Insert Data into Class Table
insert into Class(cno, lecturer, day, time, room)
values('isys155','e123','Wed','17:30','80.01.12');
insert into Class(cno, lecturer, day, time, room)
values('cosc121','e205','Thu','08:30','12.10.02');
insert into Class(cno, lecturer, day, time, room)
values('artc131','e301','Mon','10:30','10.08.09');
insert into Class(cno, lecturer, day, time, room)
values('cosc101','e205','Tue','14:30','14.09.05');

--Insert Data into Enrol table
insert into Enrol(sno, cno, grade) values('s1001', 'isys155',
'HD');
insert into Enrol(sno, cno, grade) values('s1003',
'cosc121', '');
insert into Enrol(sno, cno, grade) values('s1005', 'artc131',
'CR');
insert into Enrol(sno, cno, grade) values('s1006',
'cosc101', '');
```

2. The Schema for the Academics Database is given below:

DEPARTMENT(deptnum, descrip, instname, deptname, state, postcode)

ACADEMIC(acnum, deptnum*, famname, givename, initials, title)

PAPER(panum, title)

AUTHOR(panum*, acnum*)

FIELD(fieldnum, id, title)

INTEREST(fieldnum*, acnum*, descrip);

2.1 Explanation of the SQL Query

```
SELECT fieldnum, title
FROM field
where (fieldnum>=500 and fieldnum<=599)
                    or (upper(title) like 'DATA %'
                    or upper(title) like '% DATA %'
                    or upper(title) like '% DATA');
```

The above query extracts the attributes fieldnum, title from the field table by verifying the condition specified in the where clause by verifying the condition and if the where clause fails then the OR statements will be verified i.e., the title will be converted into uppercase statement using the upper() and search for the pattern mentioned with like(%) Operator. The query will Executes in the following order:

1. FROM -> In the first step it retrieves all the data stored in the particular table mentioned i.e data from table “field”.
2. WHERE -> In the second step it checks whether the condition specified in the where clause is satisfied or not.

   Condition1:

   It checks for the filednum must be greater than or equals to 500 and must be less than or equal to 599.

   Condition2:

   If the condition fails then it will execute the OR statements which states that the title must contain the word DATA. It again holds 3 individual conditions such as:

   (upper(title) like 'DATA %')

   The above condition converts he whole title into

uppercase

using the upper() and checks for the word DATA at the beginning of the title and can be checked with like operator.

or upper(title) like '% DATA %'

The above condition checks for the word DATA at the middle of the title and can be checked with like operator.

or upper(title) like '% DATA'

The above condition checks for the word DATA at the end of the title and can be checked with like operator.

3. **SELECT->** In the final step it selects the required fields from the given table i.e., fieldnum, title from the table “field”.

Therefore, the output consists of the titles from the fieldnum 500 to 599 and the titles which has the pattern data.

--2.2 Correction of the SQL Query

--Given SQL Query:

--Select panum, title

--From author. Interest, paper

--Where author.acnum =interest.acnum;

--The Correct Query for the given incorrect query is given
--below:

```
Select PAPER.PANUM, PAPER.TITLE, AUTHOR.ACNUM,
INTEREST.FIELDNUM

From AUTHOR, INTEREST, PAPER

Where AUTHOR.PANUM = PAPER.PANUM and
AUTHOR.ACNUM = INTEREST.ACNUM;
```

--2.3 Number of academics in the department
--where deptnum is 100 is-

```
select count(*)

from ACADEMIC

where DEPTNUM = 100;
```

--The above SQL Query gives the count of the total
--academics in the Academic table with deptnum 100.

--2.4 Listing the titles of all papers in an
--alphabetical order

```
SELECT *

FROM PAPER

ORDER BY TITLE ASC;
```

--The SELECT * statement selects all the titles from the

--PAPER table by ordering them(by using ORDER BY) in
--ascending order(ASC) thereby produces the result in an
--alphabetical order.

--2.5 Returning the Details research fields in which --titles starting with the word DATA-

```
SELECT *

FROM FIELD

where TITLE like 'Data%' AND TITLE not like 'Databases%';
```

--The above Query gives the FIELDNUM and the TITLE
--from the FIELD table in which the title must start with the
--word DATA.

--2.6 Query to list the panum, title and author --acnum of each paper-

--Query:

```
Select PAPER.PANUM, PAPER.TITLE, AUTHOR.ACNUM

From PAPER join AUTHOR on PAPER.PANUM =
AUTHOR.PANUM;
```

--The above Query gives the panum from the AUTHOR
--table and the title from the PAPER table and the ACNUM
--from the AUTHOR table using join.

```
--2.7 Give the famname and givename of
--academics working for 'RMIT CS'

Select ACADEMIC.FAMNAME, ACADEMIC.GIVENAME

From ACADEMIC, DEPARTMENT

Where (ACADEMIC.ACNUM >= 200 and
ACADEMIC.ACNUM <= 299)

AND DEPARTMENT.DESCRIP = 'RMIT CS'

ORDER BY ACADEMIC.FAMNAME,
ACADEMIC.GIVENAME ASC;

--The above query gives the famename and givename
--from ACADEMIC table from ACNUM 200 to 299 and the
--descrip must be RMIT CS in the ascending order.
```

```
--2.8 Give the famname and givenname for
--institutions in Victoria.

Select ACADEMIC.FAMNAME, ACADEMIC.GIVENAME

From ACADEMIC, DEPARTMENT

Where DEPARTMENT.STATE = 'VIC' OR
DEPARTMENT.STATE = 'Vic';

--The above query gives the famname and the givename of
--ACADEMIC which works for the state VICTORIA.
```

--2.9 Searching for the academics which do not --have titles.

```
Select FAMNAME, GIVENAME
From ACADEMIC
Where TITLE is null
ORDER BY FAMNAME, GIVENAME ASC;
```

--The above query gives the givenname and the famname --from the ACADEMIC table in which the academics have --no title in an ascending order.

--2.10 Number of Institutions in the Database

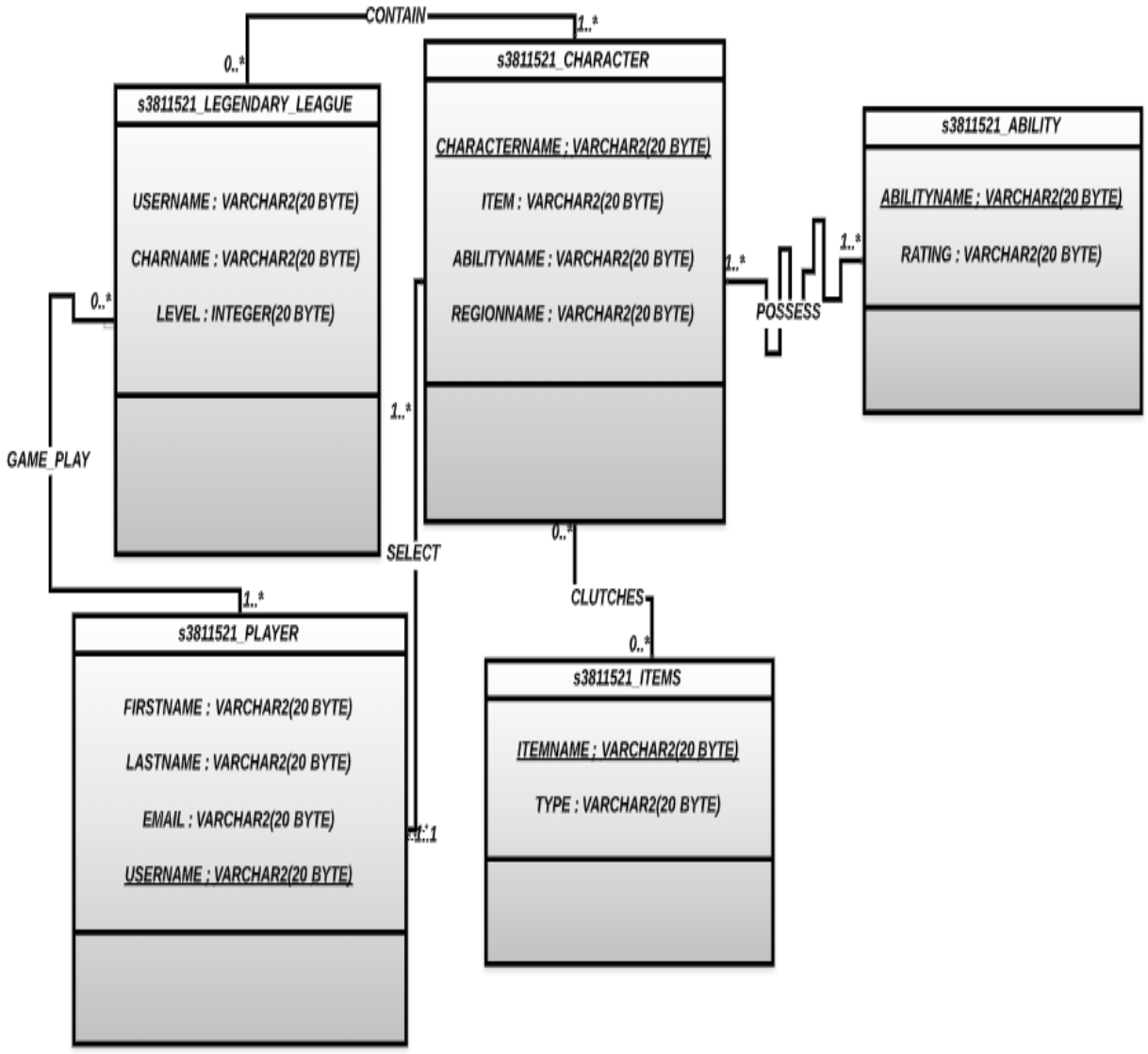
```
Select count(DISTINCT instname)
from DEPARTMENT;
```

--The above Query gives the count of the institutions in the --database and the DISTINCT keyword eliminates the --duplicates in the instname.

3. ER Diagram for the data of an Online Multiplayer Game.

It can be drawn in the following ways,

a. With the LEGENDARY_LEAGUE class containing level attribute.



In the above ER-Diagram the LEGENDARY_LEAGUE table was introduced and related to the player table with “GAME_PLAY” relation with 1 : n relationship which means the League will have 1 or more players similarly, from player to game it is 0 : n i.e., even if league exists or doesn’t exists player can create account and play the game. Player in turn can play in different leagues too.

In the same view character and Legendary League has the relation with name “CONTAIN”. A 1 to n relationship from league to character states that player must play the league games with at-least one character and can select any character he want. In the backward direction it is 0 to n which adds that even there are no leagues organised characters are still present for the players to play the game.

Each player can play with one or more characters from the Character table and the relation between player table and the Character table can be represented as “SELECT” with 1 to n relationship from player to character stating that the player must select a character in order to play the game. A 0 to n relationship in the reverse direction gives that even the player is not registered or chosen the character the game characters are present.

In the s3811521_Legendary league table –

It has three attributes namely,

--CHARNAME :

Denotes the name of the character in character table

--USERNAME :

Denotes the name of the user in player table

--LEVEL :

Denotes the level of the player in the game.

The Game can have any number of players and each player can play with any number of characters. The character may change for every game but the level of the player should remain same until the rating changes.

In the Player table—

USERNAME is unique

FIRSTNAME, LASTNAME and EMAIL are the attributes.

In the Character table—

CHARNAME is unique

ITEM, REGIONNAME, ABILITY are the attributes.

ITEM gets the ITEMNAME from the table ITEM and acts as a foreign key. Similarly ABILITY gets the ABILITYNAME from the ABILITY table.

Each Character can hold one or more Items but it is not mandatory to hold an item and hence the relationship from character to item is 0 to n, in reverse case if the relationship is 0 to n as all weapons are not required to be picked up by the character and the relation is named as “CLUTCHES”.

Similarly, in the case of ability relation is named as “POSSESS”, as character must have minimum one ability relationship is 1 to n. Vice-versa from ability to character the relationship is 1 to n.

The regionname in the Character table describes that the character must be from a particular region.

In the Item table—

ITEMNAME is unique

ITEMTYPE is an attribute.

ITEMTYPE describes the type of item that the character holds.

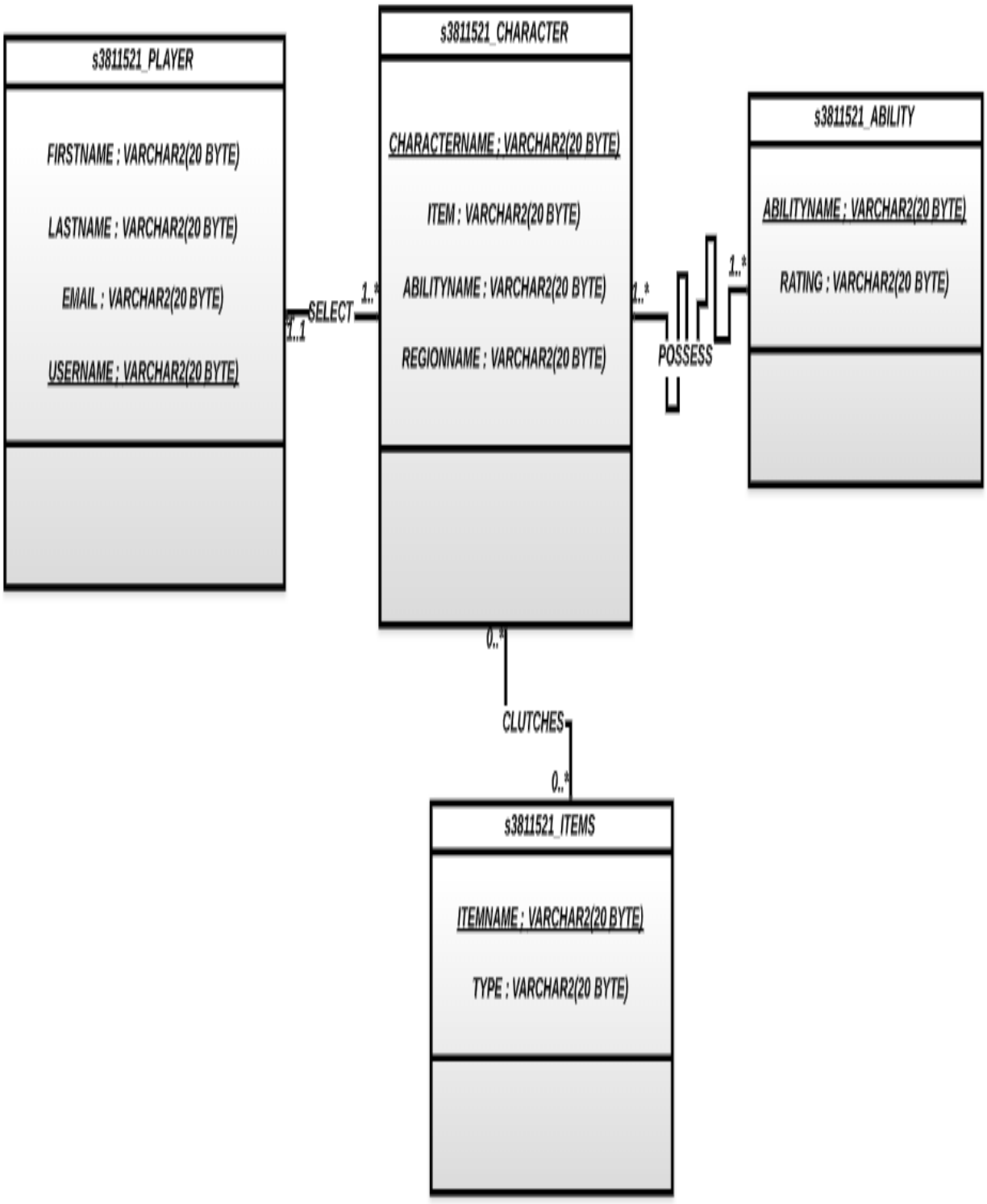
In the Ability table—

ABILITYNAME is unique.

RATING is an attribute.

RATING is the attribute that is assigned or calculated based on the performance of the player using the particular character possessing the particular ability.

b. Without the LEGENDARY_LEAGUE class.



In the above Entity Relationship diagram for the Legendary game database, Consider four tables namely Player, Character, Ability and Item.

Each player can play with one or more characters from the Character table and the relation between player table and the Character table can be represented as “SELECT” with 1 to n relationship from player to character stating that the player must select a character in order to play the game. A 0 to n relationship in the reverse direction gives that even the player is not registered or chosen the character the game characters are present.

In the Player table—

USERNAME is unique

FIRSTNAME, LASTNAME and EMAIL are the attributes.

In the Character table—

CHARNAME is unique

ITEM, REGIONNAME, ABILITY are the attributes.

Each Character can hold one or more Items but it is not mandatory to hold an item and hence the relationship from character to item is 0 to n, in reverse case if the relationship is 0 to n as all weapons are not required to be picked up by the character and the relation is named as “CLUTCHES”.

Similarly, in the case of ability relation is named as “POSSESS”, as character must have minimum one ability relationship is 1 to n. Vice-versa from ability to character the relationship is 1 to n.

The regionname in the Character table describes that the character must be from a particular region.

In the Item table—

ITEMNAME is unique

ITEMTYPE is an attribute.

In the Ability table—

ABILITYNAME is unique.

RATING is an attribute.

RATING is the attribute that is assigned or calculated based on the performance of the player using the particular character possessing the particular ability.

c. ER diagram by creating a database with tables,

```
--CREATING TABLE LEGENDARY LEAGUE:
CREATE TABLE S3811521.LEGENDARYLEAGUE (
USERNAME VARCHAR2(20 BYTE),
CHARNAME VARCHAR2(20 BYTE),
CONSTRAINT PLAYER_USERNAME_FK FOREIGN KEY (USERNAME) REFERENCES S3811521.PLAYER (USERNAME),
CONSTRAINT CHARACTER_CHARNAME_FK FOREIGN KEY (CHARNAME) REFERENCES S3811521.CHARACTER (CHARACTERNAME)
) ;
```

CONSTRAINTS --- USERNAME : FOREIGN KEY

CHARNAME : FOREIGN KEY

```
--CREATING TABLE PLAYER:
CREATE TABLE S3811521.PLAYER(
FIRSTNAME VARCHAR2(20 BYTE),
LASTNAME VARCHAR2(20 BYTE),
```

```
EMAIL VARCHAR2(20 BYTE),
USERNAME VARCHAR2(20 BYTE),
CONSTRAINT PLAYER_USERNAME_UNIQUE
UNIQUE(USERNAME)
);
```

CONSTRAINTS --- USERNAME : UNIQUE

```
--CREATING TABLE CHARACTER:
CREATE TABLE CHARACTER (
CHARACTERNAME VARCHAR2(20 BYTE),
ITEM VARCHAR2(20 BYTE),
ABILITY VARCHAR2(20 BYTE),
REGION VARCHAR2(20 BYTE),
CONSTRAINT ABILITY_FK FOREIGN KEY(ABILITY)
REFERENCES ABILITY(ABILITYNAME),
CONSTRAINT ITEM_FK FOREIGN KEY(ITEM)
REFERENCES ITEM(ITEMNAME)
CONSTRAINT CHARACTER_CHARNAME_UNIQUE
UNIQUE(CHARNAME)
);
```

CONSTRAINTS --- ABILITY : FOREIGN KEY

ITEM : FOREIGN KEY

CHARACTERNAME : UNIQUE

```
--CREATING TABLE ITEM:
CREATE TABLE S3811521.ITEM(
ITEMNAME VARCHAR2(20 BYTE) NOT NULL,
TYPE VARCHAR2(20 BYTE) NOT NULL,
CONSTRAINT ITEMNAME UNIQUE (ITEMNAME)
);
```

CONSTRAINTS --- ITEMNAME : UNIQUE, NOT NULL

TYPE : NOT NULL

```
--CREATING TABLE ABILITY:
CREATE TABLE S3811521.ABILITY(
ABILITYNAME VARCHAR2(20 BYTE),
RATING VARCHAR2(20 BYTE),
CONSTRAINT ABILITY UNIQUE (ABILITYNAME)
);
```

CONSTRAINTS --- ABILITYNAME : UNIQUE

Order for table creation is *ABILITY/ITEM, CHARACTER/PLAYER, LEGENDARY LEAGUE.*

Due to the key constraints and their dependencies.

Character table has the foreign keys from ITEM and ABILITY tables where as LEGENDARY_LEAGUE has the keys from PLAYER and CHARACTER tables.

ER diagram for the above tables is as follows: