



FISHING TOURNAMENT ORACLE 19C PROJECT

By Charlie Evert



MAY 17, 2021
SAINT JOSEPH'S UNIVERSITY MBA
DSS 630

Introduction

Fictional characters unite in this once-in-a-lifetime fishing competition. Movie characters, TV characters and even pirates squared off to see who could wrangle the largest fish over the course of a weekend. The results have been posted, but the judges do not know which boat should come in first, which baits were used, and which Fishes were ultimately the most impressive. Through the usage of SQL, it is easy to see the who, what and the why of this great occurrence. Please read on to discover the database design that I used to encapsulate this event, as well as some key queries that help to visualize the results of the event.

Database Design (Entities)

Please see Appendix A1 for the schema of this FISHING database.

The entity Fisherman is described by a unique Fisherman ID (in case of identical names) as its primary key. It is then described by its composite candidate key (FirstName, LastName) as well as other attributes including the boat they are on, and the number of fish that they caught.

The entity Boat is described by its primary key, its Boat Name. It is then described by its attributes including its captain (designated by FishermanID), its fish capacity and its fisherman capacity.

The entity Fish is defined by its unique FishID, since more than one fish may have the same name. This entity possesses the attributes of its name, its weight, the bait slice that caught it, and the ID of the fisherman that caught it.

The entity Bait is defined by BaitSliceID as its primary key and possesses the attributes of what is it (BaitType), its age in days and the ID of the fisherman that used it.

Database Design (Entity Relationships)

Please see Appendices A2 and A3 for both the dependency diagram and the entity relationship diagram of the Fishing Tournament database.

There must be only one captain per boat, and only one boat per fisherman. Since many participants are pirates, and pirates can only belong to one ship, this trigger must be enforced, or fighting may ensue. These triggers are pictured in yellow in Appendix A2.

The participating boats with at least one catch as well as the ranking of boats (by number of fish caught) are two views that help understand relative performance in this tournament. Thus, these two views have been diagrammed in red.

The following rules pertain the cardinality of each entity relationship, pictured in Appendix A3:

- Fishermen can catch 0 to many fish, and fish can be caught by 0 or 1 fisherman.
- Fish can eat zero to many pieces of bait, but bait can at most be eaten by one fish.
- Bait must be possessed by a single fisherman, but fishermen can have 0 to many bait slices.
- Fishermen can fish off 0 or one boat, but boats must have 1 to many fishermen on board.

Database Design (Table & Data Creation)

Appendices B1 through B5 demonstrate the CREATE TABLE & INSERT INTO [TABLE] VALUES statements used to create the tournament database. Primary keys and foreign keys were assigned after table creation by using the Action button under the constraints tab present when viewing each table (as described in Appendix B5). All tables are in BCNF and are connected by foreign keys as pictured under the CONSTRAINTS output picture.

Triggers (See Appendix C)

One trigger welcomes a new captain, and the other welcomes new crew. Upon inserting or updating a new captain for a boat entity, this welcome message displays. Similarly, upon inserting or updating a new boat for a fisherman, a welcome message is displayed for the new crew member. These triggers assume that Server Output is on, so code for this must be enabled on each terminal prior to fishermen updating their boats (or boat captains updating their role as captain).

Queries (See Appendix D)

The first query selects the two fishermen aboard the Black Pearl ship along with their names. Will Turner caught one more fish than Jack Sparrow. This is likely due to differences in bait - Jack Sparrow used primarily Boots, Gold and the like whereas Will Turner used actual bait. I utilized naming conventions to rename each field to be more applicable, as is pictured in Appendix D.

The second query selects the distinct count of fish caught, along with the largest fish and its weight. Someone managed to pull in a 1500 lb. Willy Wonka by using chocolate as bait! This query was made possible due to the MAX and COUNT commands, as well as the DISTINCT modifier.

Views (See Appendix E)

The first view shows the distinct count of bait types used. This view is handy so that future competitors know what sorts of bait to bring along. I chose not to include a ranking of

each bait so that no future competitor could gain a competitive edge through bringing exclusively one sort of bait; sometimes variety is the spice of life. A future DBMS administrator could pull up the ranking of each, however, given current tables and values.

The second view shows each boat along with the number of fish caught on each one. Metallica's 1 as well as Davey Jones' Locka were very effective, whereas John Locke's The Lost and Will Turner's Black Peal were less effective in the tournament. The competition ended in a tie if results were based around number of fish caught.

Stored Procedures (See Appendix F)

In my stored procedure, I used the CREATE OR REPLACE procedure command to generate a procedure to display a fisherman's name after entering their Fisherman ID. First name and last name were concatenated, and their output was displayed. I demonstrated this procedure in action by calling the fisherman ID 1, which corresponds to Count Metallica (the captain of the ship named 1). Server output must be on/enabled prior to calling this procedure.

Cursors (See Appendix G)

This cursor displays the boat names of entered data for corresponding Last Names and Fisherman IDs. It is handy in terms of looking up participants' boats without having to generate SELECT queries each time data is needed. Server output must be set to on to see any message, as is encapsulated in the DBMS output.

Conclusion

This project was very interesting, and I found that I learned more doing it than doing the homework. I did struggle quite a bit in formulating trigger, procedures, and (especially) cursors, and was thankful that I was able to get mine to run! In the future, I will experiment with more complex functions, given the relative incompleteness of the ones displayed in this project.

Despite issues, I feel that my completed project is quite thorough and demonstrates a decent understanding of Oracle 19c.

Another limitation that I discovered in inserting data. I would like to learn how to insert bulk data so that I may use a data set instead of making up data. Experimenting with 1 million or more rows per table would have been helpful had I known how to convert data to SQL. This improvement would be the greatest for me, in that I would be able to use SQL for its intended purpose. Plus, it would be great to be able to convert SQL to CSV and similar so that I may use them with BI visualization tools.

APPENDIX A1 (Schema)

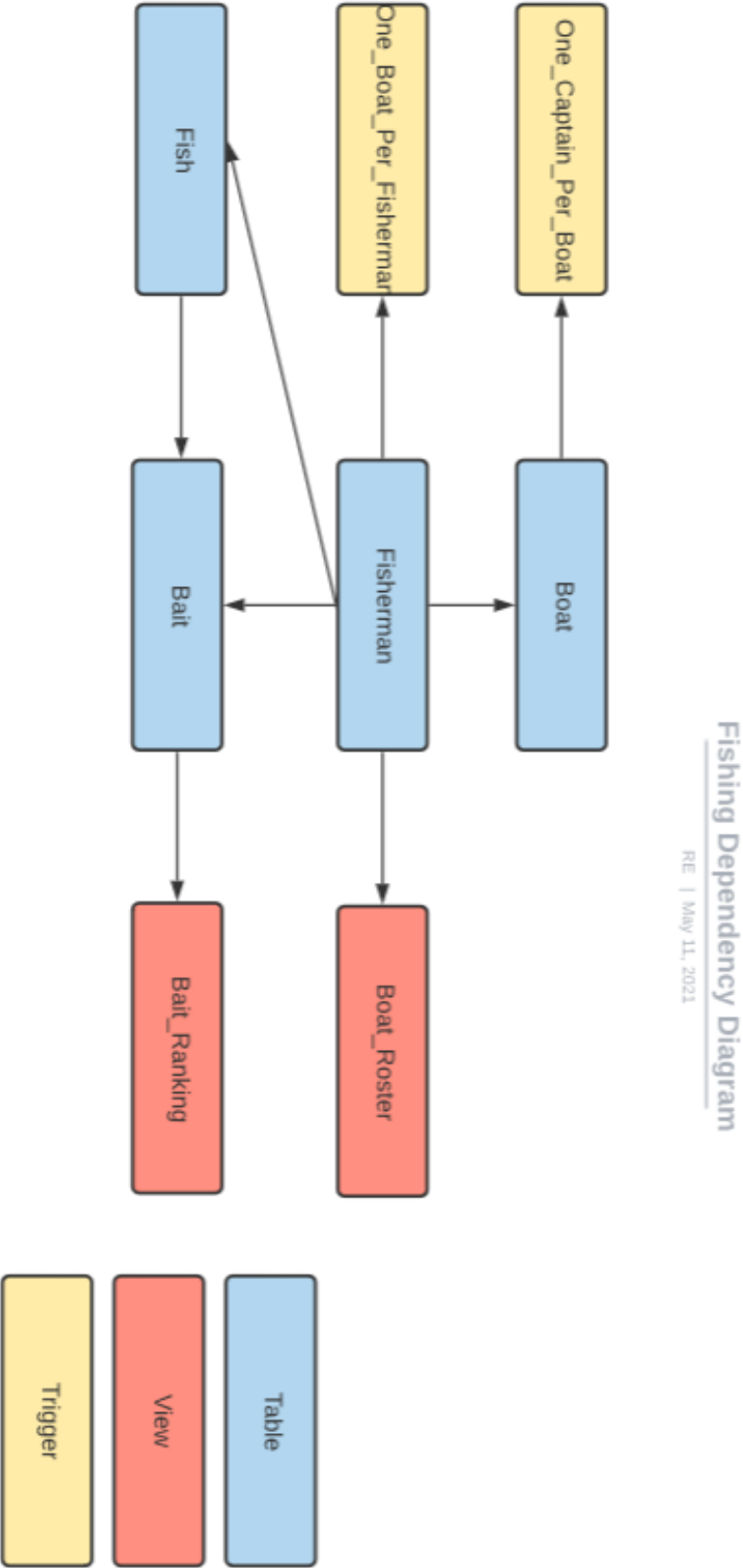
Fisherman(FishermanID, FirstName, LastName, BoatName, NumberOfFishCaught)

Boat(BoatName, FishermanID, FishCapacity, FishermanCapacity)

Fish(FishID, FishName, Weight, BaitSliceID, FishermanID)

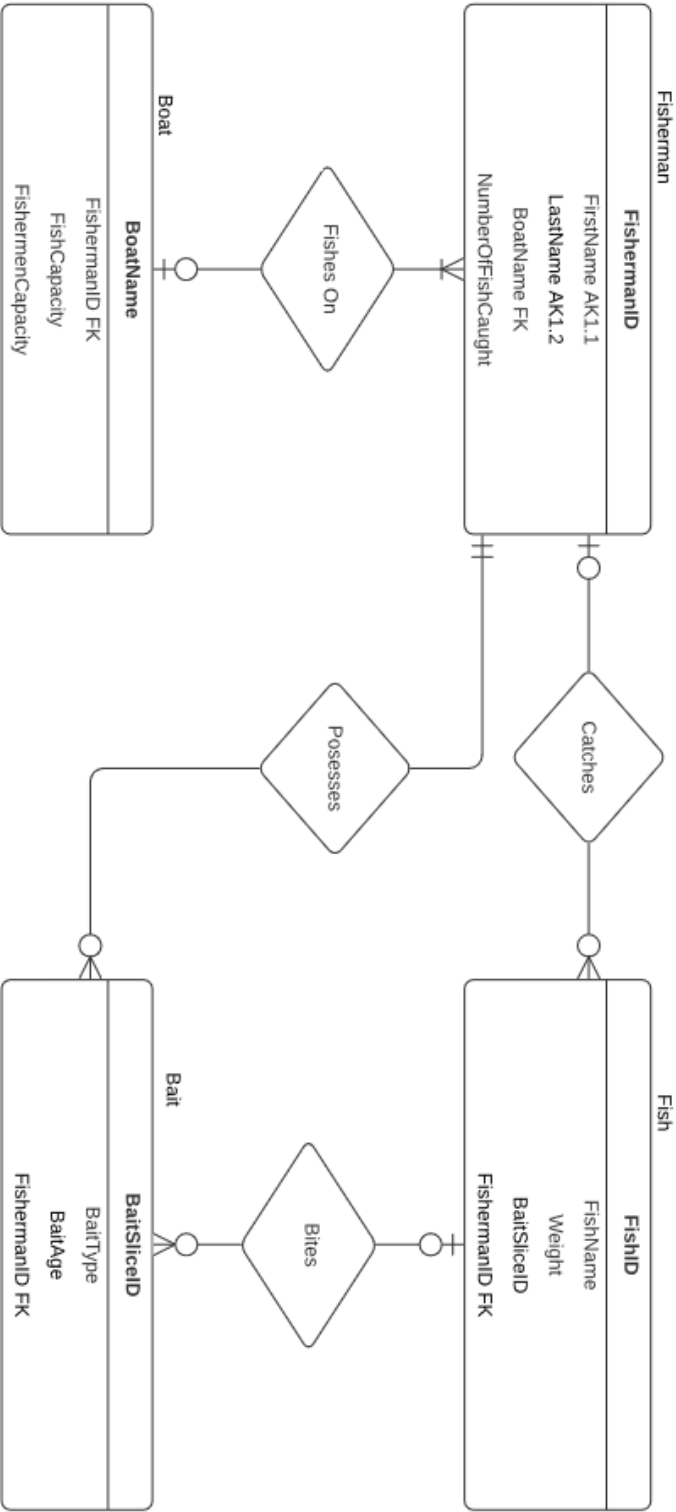
Bait(BaitSliceID, BaitType, BaitAge, FishermanID)

APPENDIX A2 (Dependency Diagram)



APPENDIX A3 (Entity Relationship Diagram)

Fishing ER Diagram



APPENDIX B1 (Fisherman Table Design)

```
CREATE TABLE Fisherman (
    FishermanID INT NOT NULL,
    FirstName VARCHAR(20),
    LastName VARCHAR(20),
    BoatName VARCHAR(20),
    NumberOfFishCaught INT NOT NULL
);
```

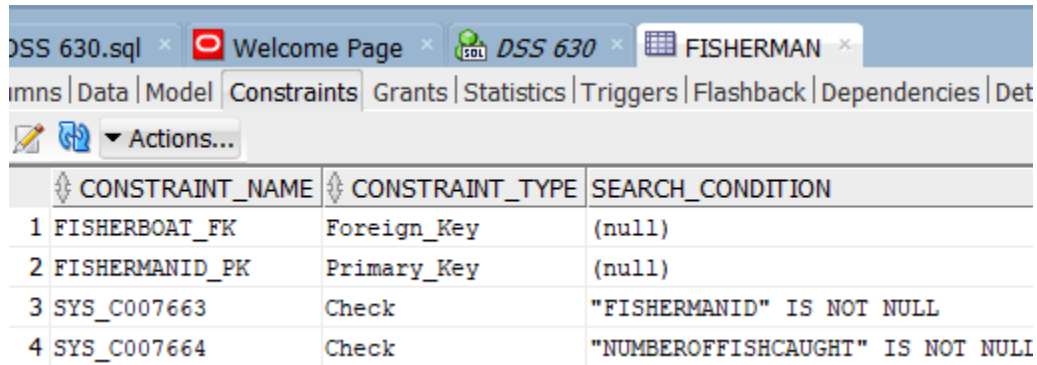
VALUES:

```
INSERT INTO Fisherman VALUES(
FishermanID, 'FirstName', 'LastName', 'BoatName', NumberOfFishCaught);
COMMIT;
```

TABLE:

FISHERMANID	FIRSTNAME	LASTNAME	BOATNAME	NUMBEROFFISHCAUGHT
1	Count	Metallica	1	2
2	Davey	Jones	DaveyJonesLocka	2
3	John	Locke	TheLost	1
4	Will	Turner	BlackPearl	1
5	Jack	Sparrow	BlackPearl	0

CONSTRAINTS:



The screenshot shows the SQL Developer interface with the 'FISHERMAN' table selected. The 'Constraints' tab is active, displaying a list of constraints for the table. The constraints are as follows:

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	FISHERBOAT_FK	Foreign_Key	(null)
2	FISHERMANID_PK	Primary_Key	(null)
3	SYS_C007663	Check	"FISHERMANID" IS NOT NULL
4	SYS_C007664	Check	"NUMBEROFFISHCAUGHT" IS NOT NULL

APPENDIX B2 (Boat Table Design)

```
CREATE TABLE Boat (
    BoatName VARCHAR(15) NOT NULL,
    FishermanID INT NOT NULL,
    FishCapacity INT NOT NULL,
    FishermanCapacity INT NOT NULL
);
```

VALUES:

```
INSERT INTO Boat VALUES(
'BoatName', FishermanID, FishCapacity, FishermanCapacity);
```

TABLE:

BOATNAME	FISHERMANID	FISHCAPACITY	FISHERMANCAPACITY
1	1	10	1
DaveyJonesLocka	2	100	10
TheLost	3	1	1
BlackPearl	4	9	2

CONSTRAINTS:

630.sql	Welcome Page	BOAT
s	Data	Model
Constraints	Grants	Statistics
Triggers	Flashback	Dependencies
Det		
Actions...		
CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
BOATFISHERMAN_FK	Foreign_Key	(null)
BOATNAME_PK	Primary_Key	(null)
SYS_C007659	Check	"BOATNAME" IS NOT NULL
SYS_C007660	Check	"FISHERMANID" IS NOT NULL
SYS_C007661	Check	"FISHCAPACITY" IS NOT NULL
SYS_C007662	Check	"FISHERMANCAPACITY" IS NOT NULL

APPENDIX B3 (Fish Table Design)

```
CREATE TABLE Fish (
    FishID INT NOT NULL,
    FishName VARCHAR(15) NOT NULL,
    Weight DECIMAL NOT NULL,
    BaitSliceID INT DEFAULT NULL,
    FishermanID INT DEFAULT NULL
);
```

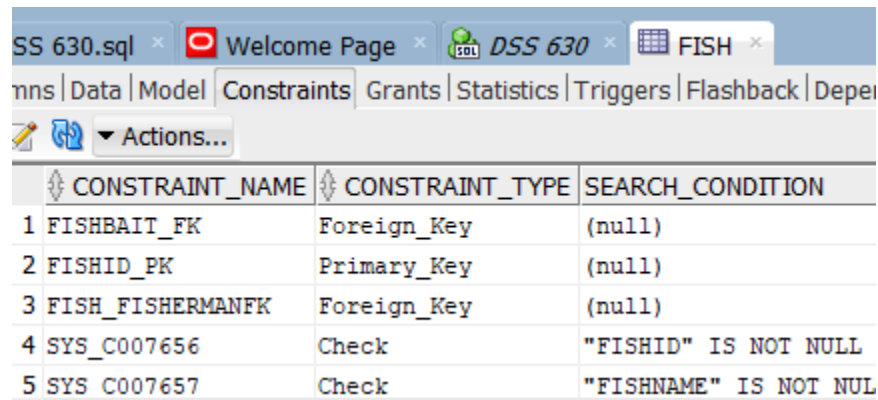
VALUES:

```
INSERT INTO FISH VALUES(
FishID, 'FishName', Weight, BaitSliceID, FishermanID);
COMMIT;
```

TABLE:

FISHID	FISHNAME	WEIGHT	BAITSLICEID	FISHERMANID
1	CheeseBurger	5	1	1
2	Mackerel	1	2	1
3	Shark	10	5	2
4	MobyDick	1000	7	2
5	WillyWonka	180	8	3
7	DaveyJones	1500	13	4

CONSTRAINTS:



The screenshot shows the SQL Developer interface with the 'FISH' table selected. The 'Constraints' tab is active, displaying a list of constraints for the table. The constraints are as follows:

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	FISHBAIT_FK	Foreign_Key	(null)
2	FISHID_PK	Primary_Key	(null)
3	FISH_FISHERMANFK	Foreign_Key	(null)
4	SYS_C007656	Check	"FISHID" IS NOT NULL
5	SYS_C007657	Check	"FISHNAME" IS NOT NUL

APPENDIX B4 (Bait Table Design)

```
CREATE TABLE Bait (
    BaitSliceID INT NOT NULL,
    BaitType VARCHAR(15) NOT NULL,
    BaitAge INT NOT NULL,
    FishermanID INT NOT NULL
);
```

VALUES:

```
INSERT INTO Bait VALUES(
    BaitSliceID, BaitType, BaitAge, FishermanID);
COMMIT;
```

TABLE:

BAITSLICEID	BAITTYPE	BAITAGE	FISHERMANID
1 Bacon		1	1
2 Bacon		99	1
3 Bacon		96	1
4 Bacon		96	1
5 Mackerel		1	2
6 Mackerel		5	2
7 Tuna		5	2
8 Chocolate		500	3
9 Chocolate		500	3
9 Chocolate		500	3
10 Chocolate		500	3
11 Squid		1	4
12 Squid		1	4
12 Squid		1	4
13 Wood		1	4
14 Boot		1	5
15 Gold		999	5
16 Nothing		999	5

CONSTRAINTS:

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1 BAITSLICEID_PK	Primary_Key	(null)
2 FEESHFK	Foreign_Key	(null)
3 SYS_C007652	Check	"BAITSLICEID" IS NOT NULL
4 SYS_C007653	Check	"BAITTYPE" IS NOT NULL
5 SYS_C007654	Check	"BAITAGE" IS NOT NULL

APPENDIX B5 (Procedure for Foreign/Primary Key Constraints)

To add Primary Keys, I clicked on the table names in the drop-down list, then clicked on the Constraints tab. Then, I clicked on Actions, Constraints and finally Add Primary Key. Then I named the Primary Keys as the Primary Key Attribute plus _PK for easy reference.

To add Foreign Keys, I clicked on the table names in the drop-down list, then clicked on the Constraints tab. Then, I clicked on Actions, Constraints and finally Add Foreign Key. Then I named the Foreign Keys as separate names plus _FK for easy reference.

To see both sets of values, please view the CONSTRAINTS section of prior B Appendices.

APPENDIX C (TRIGGERS)

1.

```
CREATE OR REPLACE TRIGGER Ahoy_New_Crew
BEFORE INSERT OR UPDATE OF BoatName ON Fisherman
REFERENCING OLD AS "OLD.BOAT" NEW AS "NEW.BOAT"
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Ahoy, a new shipmate for this ship I see. What brings you to this
new boat?');
END;
```

2.

```
CREATE OR REPLACE TRIGGER Welcome_New_Captain
BEFORE INSERT OR UPDATE OF FISHERMANID ON BOAT
REFERENCING OLD AS "OLD.CAPTAIN" NEW AS "NEW.CAPTAIN"
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Ahoy, a new captain for this ship I see.');
```

```
END;
```

APPENDIX D (Queries)**1.***INPUT:*

SELECT

```

    fisherman.numberoffishcaught AS "Fish Count of Black Pearl",
    fisherman.firstname          AS "First Name",
    fisherman.lastname           AS "Last Name"

```

FROM

```

    fisherman

```

WHERE

```

    fisherman.boatname = 'BlackPearl'

```

GROUP BY

```

    fisherman.numberoffishcaught,
    fisherman.firstname,
    fisherman.lastname

```

ORDER BY

```

    "Fish Count of Black Pearl" DESC

```

OUTPUT:

Fish Count of Black Pearl	First Name	Last Name
1	Will	Turner
0	Jack	Sparrow

2.*INPUT:*

SELECT DISTINCT

```

    COUNT(fish.fishname) AS "Fish Count",
    MAX(fish.fishname)   AS "Largest Fish Name",
    MAX(fish.weight)     AS "Largest Fish Weight"

```

FROM

```

    Fish;

```

OUTPUT:

Fish Count	Largest Fish	Largest Fish Weight
6	WillyWonka	1500

APPENDIX E (Views)

1.

INPUT:

```
CREATE VIEW Distinct_BaitTypes AS
SELECT DISTINCT(BaitType)
FROM Bait
ORDER BY BaitType DESC;
```

OUTPUT:

	BAITTYPE
1	Wood
2	Tuna
3	Squid
4	Nothing
5	Mackerel
6	Gold
7	Chocolate
8	Boat
9	Bacon

2.

INPUT:

```
CREATE VIEW BoatRoster AS
SELECT DISTINCT(BoatName), NumberOfFishCaught
FROM Fisherman
WHERE NumberOfFishCaught > 0
ORDER BY NumberOfFishCaught DESC;
```

OUTPUT:

BOATNAME	NUMBEROFFISHCAUGHT
1	2
DaveyJonesLocka	2
BlackPearl	1
TheLost	1

APPENDIX F (Stored Procedures)*INPUT:*

```

CREATE OR REPLACE PROCEDURE DISP_FISHERMAN_NAME(I_FishermanID IN
Fisherman.FishermanID%TYPE) IS
I_FirstName Fisherman.FirstName%TYPE;
I_LastName Fisherman.LastName%TYPE;

BEGIN

DBMS_OUTPUT.ENABLE;
SELECT FirstName, LastName
INTO I_FirstName, I_LastName
FROM Fisherman
WHERE FishermanID = I_FishermanId;

DBMS_OUTPUT.PUT_LINE(RTRIM(I_FirstName) || '-' || RTRIM(I_LastName));
END DISP_FISHERMAN_NAME;

TO VIEW STORE PROCEDURE (For Fisherman ID = 1)

SET SERVEROUTPUT ON
CALL DISP_FISHERMAN_NAME('1');

```

OUTPUT:

```

Count-Metallica

Call completed.

```

APPENDIX G (Cursors)

```
CREATE OR REPLACE PROCEDURE DISP_BoatName_Fisherman(I_BoatName IN
Fisherman.BoatName%TYPE) IS
I_CUSTOMERID Fisherman.FishermanID%TYPE;
I_LastName Customer.LastName%TYPE;

CURSOR BoatGroup IS
SELECT FishermanId, LastName
FROM Fisherman
WHERE BoatName = I_BoatName;
BEGIN
DBMS_OUTPUT.enable;
OPEN BoatGroup;
LOOP
    FETCH BoatGroup INTO I_FishermanID, I_LastName;
    EXIT WHEN BoatGroup%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(I_CUSTOMERID || ' ' || I_LastName);
END LOOP;

CLOSE BoatGroup;
END;
```