SQL Part 2

Kathleen Durant PhD
Northeastern University
CS3200
Lesson 6

Outline for today

- More of the SELECT command
- Review of the SET operations
- Aggregator functions
- ‘GROUP BY’ functionality
- JOIN construct (not covered in book)

Anatomy of the SELECT command

SELECT [DISTINCT] target-list FROM relation-list WHERE qualification

- **Target list** are the attributes you are **projecting** out of the relation(s) π
- **Relation-list** – name of the relations involved in the query
- **Qualification** – represents the **selection** criterion the tuples need to satisfy σ

Set operations

- **UNION** – tuple in either set
- **INTERSECT** –tuple in both sets
  - **Not supported in MYSQL**
- **EXCEPT** – tuple in the first set but not the second set
  - **Not supported in MYSQL**
- **IN** – tests membership in a set
- **EXISTS** – tests against the NULL set (correlated query)
- OPERATION **ANY** {set} – compares an attribute using an operation to all members within a set – returns true if it satisfy the condition on any member in the set
- OPERATION **ALL** {set} – compares an attribute using an operation to all members within a set – returns true if it satisfies the condition on ALL members in the set

Table Instances

- We will use these instances of the Sailors and Reserves relations in our examples.

B1

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

R1

SID	BID	DAY
22	101	10/10/96
58	103	11/12/96
31	102	01/01/01

S1

SID	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2

SID	Sname	Rating	Age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

UNION Example

- Provides **set union** functionality (no duplicate members)
- Example: You are looking for boats that are red or green

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE (B.color = 'red')

UNION

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE (B.color = 'green')

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

- UNION vs. UNION ALL
  - UNION ALL provides duplicates (multiset)

INTERSECT Example

- Returns tuples in both sets
- Example: You are looking for a red Interlake boat

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE B.color = ‘red’

INTERSECT

SELECT B2.BID, B2.Bname, B2.Color FROM Boats B2 WHERE (B2.Bname = ‘Interlake’)

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	Red

- INTERSECT NOT IN MY SQL
  - We can use a few solutions to get the INTERSECT functionality

INTERSECT Workaround Nested Query with IN

- Example: You are looking for a red Interlake boat

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE B.color = 'red' and B.BID **IN** (

SELECT B.BID FROM Boats B2 WHERE (B2.Bname = 'Interlake'))

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

- CAUTION: Make sure you are creating a set with a unique representation for a tuple
  - Create a set of primary keys
  - Still may run into trouble if you have a composite primary key

INTERSECT Workaround using Exists

- Must be a correlated query
- Example: You are looking for a red Interlake boat

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE EXISTS

(SELECT B.BID FROM Boats B2 WHERE B.Bname = 'Interlake' and B.color = 'red' and **B.BID = B2.BID**)

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Exists
F
T
F
F

- CAUTION: Make sure you are creating a set with a unique representation for a tuple
  - Equivalence of all fields in the primary key
  - Since the query is correlated no problem with composite keys

Yes we are creating an overcomplicated solution

- Example: You are looking for a red Interlake boat

```
SELECT B.BID, B.Bname, B.Color FROM Boats B
        WHERE B.color = 'red'  and  B.Bname = 'Interlake'
```

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

EXCEPT Example

- Set Difference (MINUS in Oracle)
- Example: You are looking for a red Interlake boat

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE B.color = ‘red’

EXCEPT

(SELECT bid from Boats B2 WHERE B.Bname <> ‘Interlake’)

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

BID	BName	Color
102	Interlake	red
104	Marine	red

EXCEPT

BID	BName	Color
103	Clipper	green
104	Marine	red

IN with a well known set (constants)

- Example: You are looking for a red Interlake boat

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE B.color IN ('red') and B.Bname IN ('Interlake')

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

IN with a nested query

- Example: You are looking for a red Interlake boat

SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE B.BID IN (SELECT B2.bid from Boats WHERE B.color IN ('red') and B.Bname IN ('Interlake'))

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

ANY and ALL operations

- ANY and ALL compare a single value to a set of values
- They are used with comparison operators like =, >, <, <>, >=, <=
- **Value = ANY (set)** is true if there is at least one member of the set equal to the value
- **Value = ALL (set)** is true if all members of the set are equal to the value
- What if there is no set? (The NULL set)
  - Any returns FALSE – so no return values **(Will never match value)**
  - ALL returns TRUE – so when there's nothing to match - everything is returned **(Will match every value)**
  - **Make sure you are returning a set**

ANY Example

- Example: You are looking for ALL Reservations for red boats

```
SELECT R.SID, R.BID, R.Date FROM R
        WHERE R.BID =
                ANY ( SELECT B.bid from Boats WHERE B.color IN ('red'))
```

SID	BID	DAY
22	101	10/10/96
58	103	11/12/96
31	102	01/01/01

F
F
T

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

SID	BID	DAY
31	102	01/01/01

ALL Example

- Example: Find the top rated sailor

```
SELECT S.SID, S.Sname, S.Rating, S.Age  FROM S
        WHERE S.Rating >=  ALL (SELECT S2.Rating from S2 )
```

SID	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

SID	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

SID	Sname	Rating	Age
58	Rusty	10	35.0

Aggregation functions

- Aggregate functions compute summaries of data in a table
- Most aggregate functions (all except **COUNT**) work on a single column of numeric data
- Typically want to use an alias to name the result of an aggregation function
- List of available aggregators
  - **COUNT**: The number of rows
  - **SUM**: The sum of the entries in a column
  - **AVG**: The average entry in a column
  - **MIN**, **MAX**: The minimum and maximum entries in a column

Aggregate examples

Find the average age of Sailors with a 10 rating

- SELECT AVG (S.age) FROM Sailors S WHERE S.rating=10

Find the youngest Sailor with a 10 rating

- SELECT MIN(S.age) FROM Sailors S WHERE S.rating=10

Find the number of sailors

- SELECT count(*) from Sailors

More calculations with Aggregations

- You can combine aggregate functions using arithmetic
- SELECT MAX(S.rating) as MaxRating, MIN(S.rating) as MinRating,

  MAX(S.rating) - MIN(S.rating) as Range from Sailors S;

Group By Clause

SELECT [DISTINCT] target-list FROM relation-list WHERE qualification GROUP BY grouping-list HAVING group-qualifier

- Extremely valuable clause for data exploration – allows you to stratify your data and perform aggregation
- Groups together tuples that have a common value in a field(s)
- The groupings can also be used for aggregating data
  - Allows you to apply aggregate functions to groups of rows

Changes to the Execution Policy

SELECT [DISTINCT] target-list FROM relation-list WHERE qualification GROUP BY grouping-list HAVING group-qualifier

- target-list contains attribute names, constants and terms with aggregate operations (e.g., MIN (S.age)).
- Attributes used in target-list must be in grouping-list.
- Every entry in **target-list** must be in **grouping-list**, be a constant, or be an aggregate function
- Each answer tuple corresponds to a group, and these attributes must have a single value per group. (A group is a set of tuples that have the same value for all attributes in the grouping-list.)

Conceptual Evaluation of a Query

- Compute cross-product of relation-list.
- Discard tuples that fail qualification.
- Delete `unnecessary' fields.
- Partition remaining tuples into groups by the value of attributes in grouping-list.
- Apply group-qualification to eliminate some groups.
  - Expressions in group-qualification must have a single value per group.
  - Attribute in group-qualification that is not an argument of an aggregate operation also appears in grouping-list
- One answer tuple is generated per qualifying group.

GROUP BY: Example 1

- Get the total number of boats you have by model
- SELECT Bname, COUNT(*) Inventory
- FROM Boats GROUP BY Bname

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

BName	Inventory
Interlake	2
Clipper	1
Marine	1

One row per group

GROUP BY: Example 2

- Get the total number of reservations by model
- SELECT B.Bname, COUNT(*) Reservations FROM Boats B, Reserves R WHERE B.Bid = R.Bid GROUP BY Bname

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

SID	BID	DAY
22	101	10/10/96
58	103	11/12/96
31	102	01/01/01

Bname
Interlake
Interlake
Clipper

BName	Reservations
Interlake	2
Clipper	1

GROUP BY: Example 3

- Get the model(s) of boat where you own at least 2
- SELECT B2.Bname, Inventory from (SELECT Bname, COUNT(*) Inventory FROM Boats GROUP BY Bname) B2 WHERE Inventory > 1

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

BName	Inventory
Interlake	2
Clipper	1
Marine	1

BName	Inventory
Interlake	2

HAVING CLAUSE

- Applies a selection criteria to the groups
- It can be used to select groups which satisfy a given condition
- Think of it as a WHERE clause for the returned groups

HAVING: Example 1

- Example: Find the boat models that you have at least 2 of
- SELECT Bname, COUNT(*) Inventory FROM Boats GROUP BY Bname HAVING Count(*) > 1

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

BName	Inventory
Interlake	2
Clipper	1
Marine	1

BName	Inventory
Interlake	2

HAVING EXAMPLE 2

Group the Sailors by Rating and find the age of the **youngest sailor** in each group where sailors' **age is limited to 18 to 60** years. Report on rating groups with **at least two sailors**

```
SELECT S.rating, MIN (S.age) AS
minage FROM Sailors S
   WHERE S.age >= 18 AND
              S.age <= 60
     GROUP BY S.rating
              HAVING COUNT (*) >= 2
```

Sailor

Sid	SName	Rating	Age
22	Dustin	7	45
77	Lubber	1	33
29	Brutus	8	55
31	Sparrow	10	25
32	Andy	7	35
58	Rusty	10	35
71	Zorba	9	16
64	Horatio	3	30
94	Art	3	63
95	Bob	3	25
96	Tom	7	23

HAVING EXAMPLE 2 Execution

```
SELECT S.rating, MIN (S.age)
AS minage FROM Sailors S
  WHERE S.age >= 18
  AND S.age <= 60
GROUP BY S.rating
  HAVING COUNT (*) >= 2
```

Sid	SName	Rating	Age
22	Dustin	7	45
77	Lubber	1	33
29	Brutus	8	55
31	Sparrow	10	25
32	Andy	7	35
58	Rusty	10	35
71	Zorba	9	16
64	Horatio	3	30
94	Art	3	63
95	Bob	3	25
96	Tom	7	23

Rating	Age
1.00	33
3.00	25
3.00	30
7.00	23
7.00	35
7.00	45
8.00	55
10.00	25
10.00	35

Rating	MinAge	Count(*)
1.00	33	1
3.00	25	2
7.00	23	3
8.00	55	1
10.00	25	2

Rating	MinAge
3.00	25
7.00	23
10.00	25

Subtle Differences between 'WHERE' and 'HAVING'

- **WHERE** refers to the rows of tables, and so cannot use aggregate functions
  - WHERE count(*) > 3 is ILLEGAL BUT
  - HAVING count(*) > 3 is LEGAL
- **HAVING** refers to the groups of rows - cannot use columns in the HAVING which are not in the **GROUP BY clause**
  - **SELECT MAX(rating) from Sailors WHERE rating < 5 GROUP BY age : WHERE clause has access to all fields in Sailors so the Query is LEGAL**
  - **Having clause just has access to the grouping variables** SELECT MAX(RATING) from Sailors GROUP BY age HAVING rating < 5 **Query is ILLEGAL**

ORDER BY CLAUSE

- The **ORDER BY** clause sorts the results of a query
  - You can sort in ascending (default) or descending order
  - Multiple columns and/or expressions can be given for sort but sort all columns in the same direction (all ascending or descending)
  - Found at the end of a query
  - Useful when perusing results
  - Cannot be embedded within sub-queries
- Syntax: SELECT <columns> FROM <tables> WHERE <condition> **ORDER BY <col1 [ASCENDING | DESCENDING| ASC | DESC] ,…coln [ASCENDING | DESCENDING| ASC | DESC] >**

Order by Example

- SELECT * from Sailors S ORDER BY rating;

Sid	SName	Rating	Age
22	Dustin	7	45
77	Lubber	1	33
29	Brutus	8	55
31	Sparrow	10	25
32	Andy	7	35
58	Rusty	10	35
71	Zorba	9	16
64	Horatio	3	30
94	Art	3	63
95	Bob	3	25
96	Tom	7	23

Sid	SName	Rating	Age
77	Lubber	1	33
64	Horatio	3	30
94	Art	3	63
95	Bob	3	25
22	Dustin	7	45
32	Andy	7	35
96	Tom	7	23
29	Brutus	8	55
71	Zorba	9	16
31	Sparrow	10	25
58	Rusty	10	35

Since you can sort on any field – there will be ties in the sort order

No guarantees on tie sort order

Order by Example 2

- SELECT Bname, count(*) Inventory from Boats GROUP BY Bname Order by Bname

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

BName	Inventory
Interlake	2
Clipper	1
Marine	1

BName	Inventory
Clipper	1
Interlake	2
Marine	1

- SELECT B.BID, B.Bname, B.Color FROM Boats B WHERE B.BID in (SELECT B2.bid from Boats WHERE B.color IN ('red') and B.Bname IN ('Interlake') ORDER by B2.bid) ORDER BY B.BID
  - WORKS IN MYSQL BUT NOT IN OTHER DBMS
  - POSTGRES & SQL SERVER: NO ORDER BY CLAUSE IN SUBQUERY

Other SELECT constructs: CASE

Sometimes want to group records by an attribute by a known heuristics

Use CASE to assign a value based on an attribute

```
SELECT
    CASE WHEN <CONDITIONAL> THEN VALUE
         WHEN <CONDITIONAL> THEN VALUE2
         ELSE VALUE3 END ALIASNAME , …
         FROM
```

CASE EXAMPLE

```
SELECT S.SID,
S.Sname, S.Rating,

CASE WHEN
S.RATING < 5 THEN
'BELOW'
WHEN S.RATING = 5
THEN 'MEDIAN'
WHEN S.RATING
<=10 THEN 'ABOVE'
 ELSE 'ILLEGAL
RATING' END SCALE
 FROM Sailors S;
```

Sid	SName	Rating	Age
22	Dustin	7	45
77	Lubber	1	33
29	Brutus	8	55
31	Sparrow	10	25
32	Andy	7	35
58	Rusty	10	35
71	Zorba	9	16
64	Horatio	3	30
94	Art	3	63
95	Bob	3	25
96	Tom	7	23

Sid	SName	Rating	Scale
22	Dustin	7	ABOVE
77	Lubber	1	BELOW
29	Brutus	8	ABOVE
31	Sparrow	10	ABOVE
32	Andy	7	ABOVE
58	Rusty	10	ABOVE
71	Zorba	9	ABOVE
64	Horatio	3	BELOW
94	Art	3	BELOW
95	Bob	3	BELOW
96	Tom	7	ABOVE

Other CONSTRUCTS: BETWEEN

Conditional used to check an attribute against a range of values

SELECT S.SID, S.Sname, S.Rating, S.age from Sailors S WHERE S.rating between 1 and 5;

Sid	SName	Rating	Age
22	Dustin	7	45
77	Lubber	1	33
29	Brutus	8	55
31	Sparrow	10	25
32	Andy	7	35
58	Rusty	10	35
71	Zorba	9	16
64	Horatio	3	30
94	Art	3	63
95	Bob	3	25
96	Tom	7	23

Sid	SName	Rating	Age
77	Lubber	1	33
64	Horatio	3	30
94	Art	3	63
95	Bob	3	25

OTHER SQL Quirks

- Should have been standardized 20 years ago
- SQL more consistent with the OS it lives on as opposed to itself across OS'
- Case sensitivity issue
  - Order by
  - Field = 'string', Field > 'string', Field < 'string'
  - Field like 'B%B'

Case Sensitivity in MySQL

- SQL KEYWORDS
  - **Not case sensitive**
- SQL field names, field aliases, stored routines
  - **Not case sensitive**
- SQL database names, table names, table aliases and database names and triggers
  - **– IT DEPENDS ON THE DEPLOYED FILE SYSTEM**
  - MS Windows – not case sensitive
  - MAC OS X
    - Not case sensitive for the default file system type (HFS+)
    - Case sensitive for UFS file system type
  - UNIX
    - Typically case sensitive
  - **To promote compatibility among OS's it is recommended to use lower case for these database objects**
    http://dev.mysql.com/doc/refman/5.5/en/identifier-case-sensitivity.html

Case Sensitivity for Strings

- If the field is a CHAR(N), VARCHAR(N), TEXT then will follow case sensitivity of system
- If the field is defined as a Binary type binary(n) varbinary(n)
  - CASE Sensitive comparison
- Going from case sensitive to case insensitive use collate
  - Make sure I review your setup
- Going from case insensitive to case sensitive use binary
  - SELECT bname from boats WHERE binary color like ‘Red‘

JOINS: Not fully covered in Book

- The keyword JOIN was introduced to SQL in SQL ’92
- Allows one to explicitly specify a JOIN should take place between two tables
  - Column delimited list generates a cross product
- Allows one to explicitly specify the columns that should be used in the JOIN (as opposed to a conditional)
- SELECT <FIELDS> FROM table_name1 [INNER|LEFT OUTER| RIGHT OUTER| FULL OUTER] JOIN table_name2 ON table_name1.column_name = table_name2.column_name

JOINS

- INNER JOIN: This will only return rows when there is at least one row in both tables that match the join condition.
  - Equi-join : performs an INNER JOIN using the equality operator
    - Most common JOIN using foreign keys
  - NATURAL JOIN: Equi-join performs an INNER JOIN using equality based on the fields that have common names between the two tables
    - Should typically not be used since dependent on tables' schemas
    - Returns pairs of rows with common values for identically named columns and without duplicating columns
  - returns pairs of rows satisfying a condition
- Outer Joins
  - LEFT OUTER JOIN (or LEFT JOIN): This will return rows that have data in the left table (left of the JOIN keyword), even if there's no matching rows in the right table.
  - RIGHT OUTER JOIN (or RIGHT JOIN): This will return rows that have data in the right table (right of the JOIN keyword), even if there's no matching rows in the left table.
  - FULL OUTER JOIN (or FULL JOIN): This will return all rows, as long as there's matching data in one of the tables.
- CROSS JOIN: A CROSS PRODUCT OF BOTH TABLES

INNER JOIN: BOTH TABLES CONTRIBUTE

- Want to match records from one table with a record from the other table based on a criterion
  - If the criterion is not TRUE no record in the result set
- Typical JOIN performed by most queries is an INNER JOIN based on equality
- SELECT <field-list> from T1 **JOIN** T2 **ON T1.FIELD = T2.field**

INNER JOIN EXAMPLE

Select Sailor Ids who have reserved Boat 103

- SELECT S.sname from sailors S **JOIN** Reserves R **ON** S.sid = R.sid WHERE R.bid = 103
- Same QUERY as Lecture 5 Slide 38
- SELECT S.sname from sailors S, Reserves R where S.sid = R.sid and R.bid = 103
- Benefits:
  - Explicitly specify the tables in the JOIN
  - Explicitly specify the fields used for the JOIN
  - Leads to more legible code
  - Typically leads to more efficient query plan

INNER JOIN EXECUTION

Select Sailor and reservation date for all reservations

- SELECT S.sname from Sailors S **JOIN** Reserves R **ON** S.sid = R.sid

SID	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
77	Lubber	1	33.0
29	Brutus	8	55.5

SID	BID	DAY
22	101	10/10/96
58	103	11/12/96
31	102	01/01/01
22	101	01/10/01

SName	Date
Dustin	10/10/96
Rusty	11/12/96
Lubber	01/01/01
Dustin	01/10/01

Outer JOIN

- Sometimes you want to define the number of records in your resultant set
  - Do not want to drop records just because there is not a corresponding record in another table
    - One table is Optional and another table is Mandatory
  - Outer JOIN construct allows you to set it to the number of records in one or both of the tables
  - New functionality not provided via an implicit Join
- Three different variations
  - LEFT OUTER JOIN – table to the left of the JOIN decides resultant set
  - RIGHT OUTER JOIN – table to the right of the JOIN decides resultant set
  - FULL OUTER JOIN – one record in result set for each record in the left and the right
    - Returns all pairs of rows from A and B
    - Number of records between MAX(count(A), count(B)) to (M+N)

LEFT OUTER JOIN EXAMPLE

Select Sailor Ids who have Reserved Boats

- SELECT S.sname, B.name from sailors S LEFT JOIN Reserves R ON S.sid = R.sid
- Use LEFT JOIN since you want to report on ALL Sailors

LEFT OUTER JOIN EXECUTION

Select Sailor Ids, day who have reserved boats

- SELECT S.sname from sailors S LEFT OUTER JOIN Reserves R ON S.sid = R.sid

SID	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
77	Lubber	1	33.0
29	Brutus	8	55.5

SID	BID	DAY
22	101	10/10/96
58	103	11/12/96
31	102	01/01/01
22	101	01/10/01

SName	Date
Dustin	10/10/96
Rusty	11/12/96
Lubber	01/01/01
Dustin	01/10/01
Brutus	NULL

RIGHT OUTER JOIN EXAMPLE

Select Sailor Ids who have reserved Boats

- SELECT R.Bid, B.Name from Reserves R RIGHT JOIN Boats B ON R.bid = S.bid
- This example, use RIGHT JOIN to report on ALL reservations as well as boats that have never had a reservation

RIGHT OUTER JOIN EXECUTION

Select boats that have been reserved and the sailors who have reserved them and the boats that have never been reserved

- SELECT S.SID, B.Name, R.DAY from Reserves R RIGHT OUTER JOIN Boats B ON R.bid = S.bid

SID	BID	DAY
22	101	10/10/96
58	103	11/12/96
31	102	01/01/01
22	101	01/10/01

BID	BName	Color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

SID	BName	DAY
22	Interlake	10/10/96
22	Interlake	01/10/01
31	Interlake	01/01/01
58	Clipper	11/12/96
NULL	Marine	NULL

FULL OUTER JOIN EXAMPLE

Select Sailors and the boats they have reserved – want to see sailors who have never made a reservation as well as reservations not assigned to a Sailor

SELECT S.sname from sailors S FULL OUTER JOIN Reserves R

ON S.sid = R.sid

FULL OUTER JOIN EXECUTION

Select Sailors that have reserved boats and Sailors that have never reserved boats as well as boats that have never been reserved

- SELECT X.SID, B.BId, X.DAY from BOATS B FULL OUTER JOIN (SELECT * FROM Reserves R FULL OUTER JOIN Sailors S ON R.bid = S.bid) X
- ON B.BID = X.BID

SID	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
77	Lubber	1	33.0
29	Brutus	8	55.5

SID	BID	DAY
22	101	10/10/96
22	102	01/10/01
31	102	01/01/01
58	103	11/12/96
77	NULL	NULL
29	NULL	NULL
NULL	104	NULL

SID	BID	DAY	BID	BName	Color
22	101	10/10/96	101	Interlake	blue
58	103	11/12/96	102	Interlake	red
31	102	01/01/01	103	Clipper	green
22	101	01/10/01	104	Marine	red

Summary

- SQL is a rich language that provides many ways to script a query
- Provides set function operations
- Selection qualifications
- Aggregation functions
- Grouping qualifications
  - Qualifications on the groups