Introduction to SQL, the Structured Query Language

Some slide content courtesy of Tova Milo

Outline

- *Data* in SQL
- Simple Queries in SQL
- Queries with more than one relation

Recommended reading:

Chapter 3, "Simple Queries" from **SQL for Web Nerds,** by Philip Greenspun
http://philip.greenspun.com/sql/

SQL Introduction

Standard language for querying and manipulating data

Structured **Q**uery **L**anguage

Many standards out there:

- ANSI SQL
- SQL92 (a.k.a. SQL2)
- SQL99 (a.k.a. SQL3)
- Vendors support various subsets of these
- What we discuss is common to all of them

SQL

- Data Definition Language (DDL)
  - Create/alter/delete tables and their attributes
- Data Manipulation Language (DML)
  - Query one or more tables
  - Insert/delete/modify tuples in tables

Data in SQL

1. Atomic types, a.k.a. data types
2. Tables built from atomic types

Unlike XML, no nested tables, only flat tables are allowed!

- We will see later how to decompose complex structures into multiple flat tables

Data Types in SQL

- Characters:
  - CHAR(n) -- fixed length
  - VARCHAR2(n) -- variable length
- Numbers:
  - INTEGER -- integers
  - NUMBER -- real numbers
- Times and dates:
  - DATE
  - TIMESTAMP

Tables in SQL

Table name

Attribute names

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Tuples or rows

Tables Explained

- A tuple = a record
  - Restriction: all attributes are of atomic type
- A table = a set of tuples
  - Like a list…
  - …but it is unordered: no **first()**, no **next()**, no **last()**.

Tables Explained

- The *schema* of a table is the table name and its attributes:

Product(PName, Price, Category, Manfacturer)

- A *key* is an attribute (or a set of attributes) whose value is unique
  - We underline the elements of the key

Product(PName, Price, Category, Manfacturer)

Basic SQL Query

SELECT [DISTINCT] {T_1.*attrib*, …, T_2.*attrib*}
FROM {*relation*} T_1, {*relation*} T_2, …
WHERE {*predicates*}

target-list

relation-list

qualification

- Returns (multi)sets of items satisfying conditions
- Let's do some examples…
  - Faculty ids
  - Course IDs for courses with students expecting a "C"
  - Courses taken by Jill

Basic SQL Query

SELECT [DISTINCT] {T_1.*attrib*, …, T_2.*attrib*}
FROM {*relation*} T_1, {*relation*} T_2, …
WHERE {*predicates*}

target-list

relation-list

qualification

- *target-list* A list of attributes of output relations in *relation-list*
- *relation-list* A list of relation names (possibly with a *range-variable* after each name)

  e.g. Sailors S, Reserves R
- *Qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of <, >, <=, >=, =, <>) combined using AND, OR and NOT.

What's contained in an SQL Query?

SELECT [DISTINCT] {T_1.*attrib*, …, T_2.*attrib*}
FROM {*relation*} T_1, {*relation*} T_2, …
WHERE {*predicates*}

Every SQL Query must have:

- SELECT clause: specifies columns to be retained in result
- FROM clause: specifies a cross-product of tables

WHERE clause (optional) - specifies selection conditions on the tables mentioned in the FROM clause

DISTINCT (Optional) – Indicates the resulting answer should not have duplicates

General SQL Evaluation Strategy

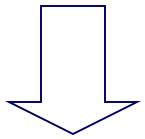
- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
- This strategy is probably the least efficient way to compute a query
- An optimizer will find more efficient strategies to compute *the same answers*.

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```



PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

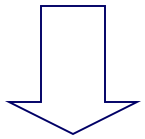
"selection"

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  PName, Price, Manufacturer
FROM    Product
WHERE   Price > 100
```



"selection" and "projection"

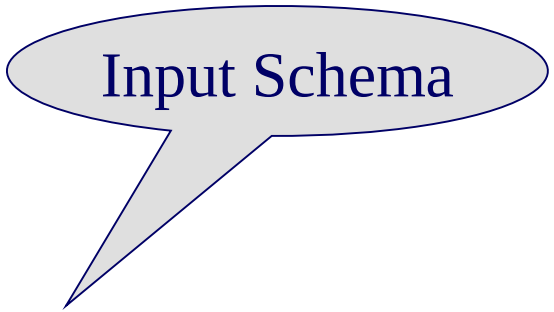
PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

Some Nice Shortcuts

- SELECT *
  - All STUDENTs
- AS
  - As a "range variable" (tuple variable): optional
  - As an attribute rename operator

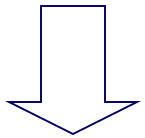
- Example:
  - Which students (names) have taken more than one course from the same professor?

A Notation for SQL Queries

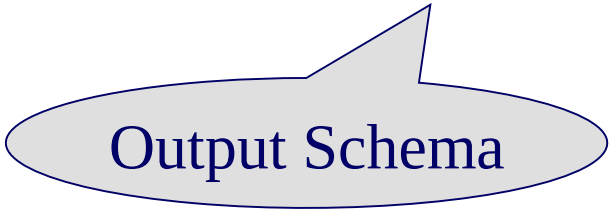


Product(PName, Price, Category, Manfacturer)

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```



Answer(PName, Price, Manfacturer)



Selections

What goes in the WHERE clause:

- x = y, x < y, x <= y, etc
  - For number, they have the usual meanings
  - For CHAR and VARCHAR: lexicographic ordering
    - Expected conversion between CHAR and VARCHAR
  - For dates and times, what you expect...
- Pattern matching on strings...

Expressions in SQL

- Can do computation over scalars (int, real or string) in the select-list or the qualification
  - Show all student IDs decremented by 1
- Strings:
  - Fixed (CHAR(x)) or variable length (VARCHAR(x))
  - Use single quotes: 'A string'
  - Special comparison operator: LIKE
  - Not equal: <>
- Typecasting:
  - CAST(S.sid AS VARCHAR(255))

The LIKE operator

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

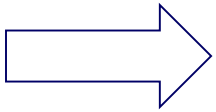
Product(PName, Price, Category, Manufacturer)

Find all products whose name mentions 'gizmo':

```
SELECT  *
FROM    Products
WHERE   PName LIKE '%gizmo%'
```

Eliminating Duplicates

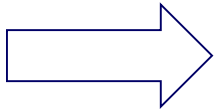
```
SELECT  DISTINCT category
FROM    Product
```



Category
Gadgets
Photography
Household

Compare to:

```
SELECT  category
FROM    Product
```



Category
Gadgets
Gadgets
Photography
Household

What happens if more attributes are selected?

Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category='Gadgets' AND price > $10.00
ORDER BY  manufacturer, price
```

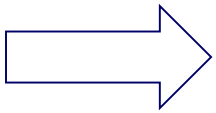
Ordering is ascending, unless you specify the DESC keyword.

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category='Gadgets' AND price > $10.00
ORDER BY manufacturer, price
```

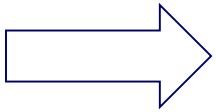
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi



?

Ordering the Results

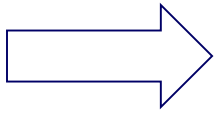
```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```



Category
Gadgets
Household
Photography

Compare to:

```
SELECT   category
FROM     Product
ORDER BY pname
```



?

Joins in SQL

- Connect two or more tables:

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

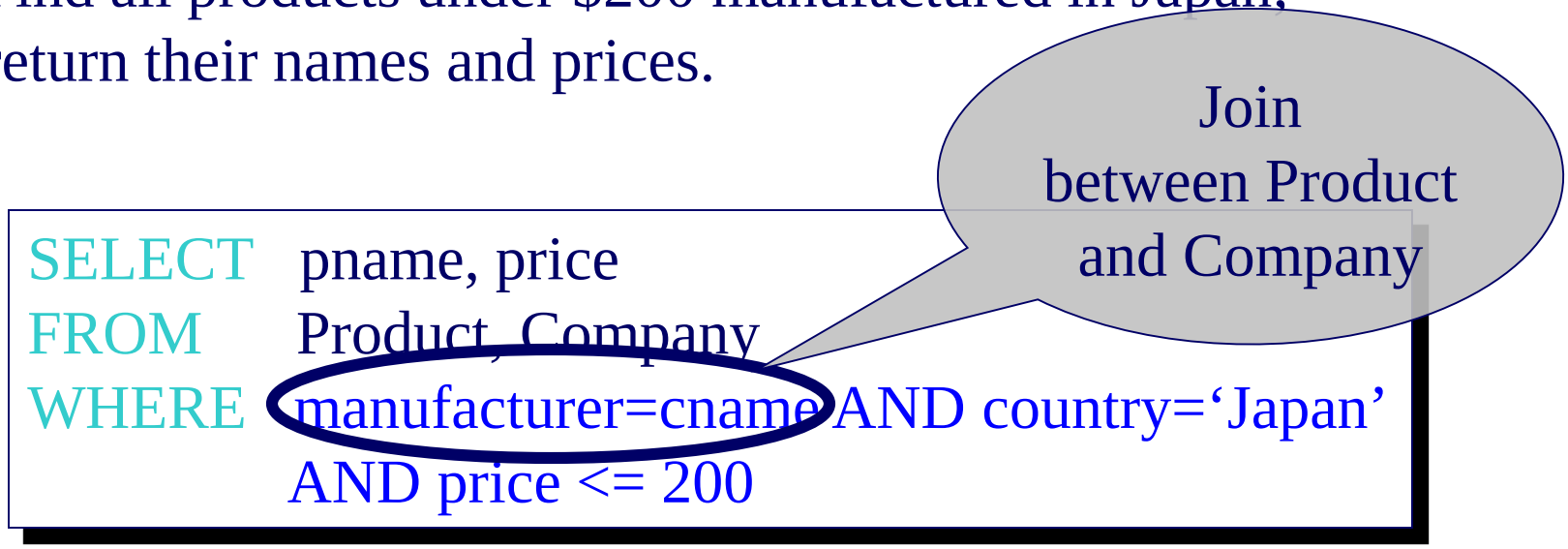
Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is the connection between them ?

Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan; return their names and prices.



Joins in SQL

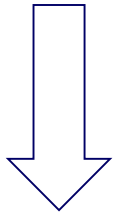
Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   pname, price
FROM     Product, Company
WHERE    manufacturer=cname AND country='Japan'
         AND price <= 200
```



PName	Price
SingleTouch	$149.99

Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all countries that manufacture some product in the 'Gadgets' category.

```
SELECT   country
FROM     Product, Company
WHERE    manufacturer=cname AND category='Gadgets'
```

Joins in SQL

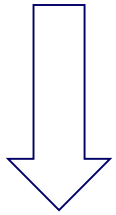
Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   country
FROM     Product, Company
WHERE    manufacturer=cname AND category='Gadgets'
```



Country
??
??

What is the problem ? What's the solution ?

Joins

Product (pname, price, category, manufacturer)
Purchase (buyer, seller, store, product)
Person(persname, phoneNumber, city)

Find names of people living in Seattle that bought some product in the 'Gadgets' category, and the names of the stores they bought such product from

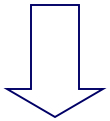
```
SELECT   DISTINCT persname, store
FROM     Person, Purchase, Product
WHERE    persname=buyer AND product = pname AND
         city='Seattle'  AND category='Gadgets'
```

Disambiguating Attributes

- Sometimes two relations have the same attribute:
  Person(pname, address, worksfor)
  Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

Which address ?



```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company
WHERE    Person.worksfor = Company.cname
```

Tuple Variables

General rule: tuple variables are introduced automatically by the system:

Product (name, price, category, manufacturer)

```
SELECT  name
FROM    Product
WHERE   price > 100
```

Becomes:

```
SELECT Product.name
FROM   Product AS Product
WHERE Product.price > 100
```

Doesn't work when Product occurs more than once:
In that case the user needs to define variables explicitly.

Table Definitions

Will be using the following relations in our examples:

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Sailors

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	*bid*	*day*
22	101	10/10/04
22	102	10/10/04
22	103	10/08/04
22	104	10/07/04
31	102	11/10/04
31	103	11/06/04
31	104	11/12/04
64	101	09/05/04
64	102	09/08/04
74	103	09/08/04

Boats

bid	*bname*	*Color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Simple SQL Query

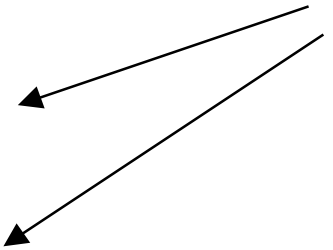
Find the names and ages of all sailors

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Result of Previous Query

SELECT S.sname, S.age
FROM Sailors S;

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Horatio	35.0
Art	25.5
Bob	63.5



Duplicate Results

Removing Duplicate Tuples

SELECT DISTINT S.sname, S.age
FROM Sailors S;

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Art	25.5
Bob	63.5

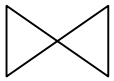
Example SQL Query…I

Find the names of sailors who have reserved boat 103

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103;
```

Result of Previous Query

sid	*bid*	*day*
22	103	10/08/04
31	103	11/06/04
74	103	09/08/04



sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Result:

sname
Dustin
Lubber
Horatio

A Note on Range Variables

- Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103;
```

OR

```
SELECT sname
FROM   Sailors, Reserves
WHERE  Sailors.sid=Reserves.sid AND
bid=103;
```

However, it is a good style to always use range variables!

Example SQL Query…2

*Find the **sids** of sailors who have reserved a red boat*

Example SQL Query…3

*Find the **names** of sailors who have reserved a red boat*

Example SQL Query…4

*Find the **colors** of boats reserved by 'Lubber'*

Example SQL Query…5

*Find the **names** of sailors who have reserved **at least** one boat*

Expressions and Strings

- AS and = are two ways to name fields in result.

- LIKE is used for string matching. '_' stands for exactly one arbitrary character and '%' stands for 0 or more arbitrary characters.

Expressions and Strings Example

Find triples (of ages of sailors and two fields defined by expressions, i.e. current age-1 and twice the current age) for sailors whose names begin and end with B and contain at least three characters.

```
SELECT  S.age, age1=S.age-1, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B';
```

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Result:

age	*age1*	*age2*
63.5	62.5	127.0

UNION, INTERSECT, EXCEPT

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- EXCEPT: Can be used to compute the set- difference operation on two *union-compatible* sets of tuples (Note: In ORACLE, the command for set-difference is *MINUS*).
- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.

Illustration of UNION…I

*Find the names of sailors who have reserved a red **or** a green boat*

Intuitively, we would write:

```
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND (B.color= 'red'  OR B.color= 'green' );
```

Illustration of UNION…2

We can also do this using a UNION keyword:

```
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color= ‘red’
UNION
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color= ‘green’ ;
```

Unlike other operations, UNION eliminates duplicates! Same as INTERSECT, EXCEPT. To retain duplicates, use

“UNION ALL”

Illustration of INTERSECT…

*Find the names of sailors who have reserved a red **and** a green boat*

We can also do this using a INTERSECT keyword:

```
SELECT  S.sname
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color= ‘red’
INTERSECT
SELECT  S2.sname
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color= ‘green’ ;
```

(Is this correct??)

(Semi-)Correct SQL Queryfor the Previous Example

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color= ‘red’
INTERSECT
SELECT  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
            AND B2.color= ‘green’ ;
```

(This time we have actually extracted the ***sids*** of sailors, and not their names.)
(But the query asks for the names of the sailors.)

Illustration of EXCEPT

*Find the sids of all sailors who have reserved red boats **but not** green boats:*

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color= ‘red’
EXCEPT
SELECT  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color= ‘green’ ;
```

Use MINUS instead of EXCEPT in Oracle

Nested Queries

- A **nested** query is a query that has another query embedded within it; this embedded query is called the **subquery**.
- Subqueries generally occur within the WHERE clause (but can also appear within the FROM and HAVING clauses)
- Nested queries are a very powerful feature of SQL. They help us write short and efficient queries.

(Think of nested **for** loops in C++. Nested queries in SQL are similar)

Nested Query 1

Find names of sailors who have reserved boat 103

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  ( SELECT  R.sid
                   FROM  Reserves R
                   WHERE  R.bid=103);
```

Nested Query 2

*Find names of sailors who **have not** reserved boat 103*

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid NOT IN  ( SELECT  R.sid
                       FROM  Reserves R
                       WHERE  R.bid=103 )
```

Nested Query 3

Find the names of sailors who have reserved a red boat

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid IN (SELECT B.bid
                                   FROM Boats B
                                   WHERE B.color = 'red'));
```

What about *Find the names of sailors who have NOT reserved a red boat?*

Revisit a previous query

*Find names of sailors who've reserved a red **and** a green boat*

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color= 'red'
INTERSECT
SELECT  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
            AND B2.color= 'green' ;
```

Revisit a previous query

*Find names of sailors who've reserved a red **and** a green boat*

```
SELECT  S.sname FROM Sailor S
WHERE S.sid IN (SELECT R.sid
                FROM  Boats B, Reserves R
                WHERE  R.bid=B.bid AND B.color='red'
                INTERSECT
                SELECT  R2.sid
                FROM  Boats B2, Reserves R2
                WHERE  R2.bid=B2.bid AND B2.color='green');
```

Correlated Nested Queries…I

- Thus far, we have seen nested queries where the inner subquery is independent of the outer query.

- We can make the inner subquery **depend** on the outer query. This is called correlation.

Correlated Nested Queries…2

Find names of sailors who have reserved boat 103

```
SELECT  S.sname
FROM  Sailors S
WHERE  EXISTS  (SELECT  *
                FROM  Reserves R
                WHERE  R.bid=103 AND R.sid=S.sid);
```

Tests whether the set is nonempty. If it is, then return TRUE.

(For finding sailors who have **not** reserved boat 103, we would use NOT EXISTS)

ANY and ALL operators

Find sailors whose rating is better than some sailor named Horatio

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.sname= 'Horatio' );
```

Using ALL operator

*Find sailors whose rating is better than **every** sailor named Horatio*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ALL(SELECT S2.rating
                     FROM Sailors S2
                     WHERE S2.sname= 'Horatio' );
```

Aggregate Operators

- What is aggregation?
  - Computing arithmetic expressions, such as **Minimum** or **Maximum**

- The aggregate operators supported by SQL are:

  COUNT, SUM, AVG, MIN, MAX

Aggregate Operators

- **COUNT**(A): The number of values in the column A
- **SUM**(A): The sum of all values in column A
- **AVG**(A): The average of all values in column A
- **MAX**(A): The maximum value in column A
- **MIN**(A): The minimum value in column A

(We can use DISTINCT with COUNT, SUM and AVG to compute only over non-duplicated columns)

Using the COUNT operator

Count the number of sailors

```
SELECT  COUNT (*)
FROM  Sailors S;
```

Example of SUM operator

Find the sum of ages of all sailors with a rating of 10

```
SELECT  SUM (S.age)
FROM  Sailors S
WHERE  S.rating=10;
```

Example of AVG operator

Find the average age of all sailors with rating 10

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

Example of MAX operator

Find the name and age of the oldest sailor

```
SELECT S.sname, MAX(S.age)
FROM Sailors S;
```

But this is illegal in SQL!!

Correct SQL Query for MAX

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age = ( SELECT MAX(S2.age)
                FROM Sailors S2 );
```

Another Aggregate Query

Count the number of different sailors

```
SELECT COUNT (DISTINCT S.sname)
FROM Sailors S
```

More tocome…

- BETWEEN…AND

Advanced SQL concepts :

- GROUP BY
- ORDER BY
- HAVING