LAB 6 Notes: SQL

- Any questions on the project (Discuss)?
- We will continue our discussion on SQL
- We will discuss the java program

SQL (Structured Query Language)

- Widely used relational database language
- Current ANSI/ISO standard is SQL99 but SQL92 is most widely used
- SQL – Query Language **but has several other aspects**
  - **DDL** (Definition Language) Create/delete/Alter tables & Views. Creating indexes/ deleting indexes
  - **DML** (Manipulation Language) Insert/Delete/ Update Rows
  - **Triggers** SQL99 supports triggers which are actions
    Triggers are not constrains
    ```
    CREATE TABLE products (
            product_no integer,
            name text,
            price numeric CHECK (price > 0));

    CREATE TRIGGER if_dist_exists
       BEFORE INSERT OR UPDATE ON products FOR EACH ROW
       EXECUTE PROCEDURE
              sendemail2managers ('did', 'distributors', 'did');
    ```
  - **Embedded and Dynamic SQL (will be covered as part of the project)**
    Allows SQL code to be executed from a host language such as C or Java.
  - **Security. (chapter 21)**
    ```
    GRANT SELECT ON products to Cashiers;
    ```
  - **Advanced Features.**
    SQL99 supports advanced features like text and XML data management
    ```
    GRANT SELECT ON products to Cashiers;
    ```

A) SQL BASIC QUERY BLOCK

```
SELECT [DISTINCT] select_list
FROM from_list
WHERE qualification;
```

Sailors(sid, name, rating, age)
```
SELECT DISTINCT name, age
FROM Sailors;
```

Selects all the distinct pairs
i.e. Chris, 20, Chris, 35

Begins and starts with B and has at least three characters)

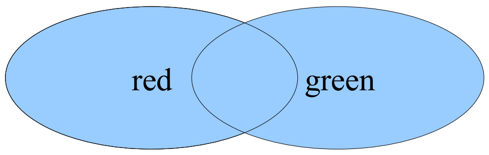
```
SELECT *
FROM Sailors
WHERE name LIKE 'B_%B'
```

B) Set Manipulation constructs: **SQL UNION, INTERSECT AND EXCEPT**

+ **Set Manipulation constructs** extend the basic query form
+ Union compatible

```
(SELECT [DISTINCT] select_list
FROM from_list
WHERE qualification)
UNION/INTERSECT/EXCEPT (MINUS)
(SELECT [DISTINCT] select_list
FROM from_list
WHERE qualification)
```

Sailors who reserved Red or green boat

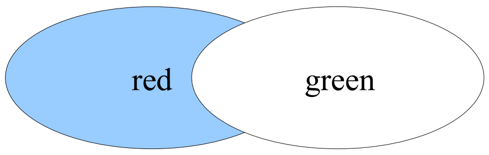


```
SELECT *
FROM SailorsReserveBoats
WHERE color=red
UNION
SELECT *
FROM SailorsReserveBoats
WHERE color=green
```

OR:

```
SELECT *
FROM SailorsReserveBoats
WHERE color=red OR color=green
```

Sailors who reserved Red but not green boat



```
SELECT *
FROM SailorsReserveBoats
WHERE color=red
EXCEPT
SELECT *
```

```
FROM SailorsReserveBoats
WHERE color=green
```

C) Set Manipulation constructs: **Correlated Nested and nested IN, EXIST**

```
(SELECT [DISTINCT] select_list
FROM from_list
WHERE attribute [NOT] IN
        (SELECT attribute
        FROM from_list
        WHERE condition)
```

Example:
NOT CORELLATED IN (work well by optimizer)

: Select sailors who reserved boat 103

```
SELECT *
FROM EMPLOYEE
WHERE sid IN
        (SELECT R.sid
        FROM RESERVES R)
```

OR:

```
SELECT *
FROM EMPLOYEE E, RESERVES R
WHERE E.sid = R.sid AND
R.bid=103;
```

CORELLATED EXISTS (ARE NOT optimized adequately)
Allows us to check whether a set is empty or not. e.g. usually helpful in correlated queries.

```
(SELECT [DISTINCT] select_list
FROM from_list
WHERE EXISTS
        (SELECT attribute
        FROM from_list
        WHERE condition)
```

e.g. select the employees with the highest salary

```
SELECT *
FROM EMPLOYEE E1
WHERE EXISTS
        (SELECT MAX(E2.salary)
        FROM EMPLOYEE E2
        WHERE E2.id = E.id)
```

OR:

```
SELECT *
FROM EMPLOYEE
WHERE E.salary=
        (SELECT MAX(salary)
        FROM EMPLOYEE);
```

D) AGGREGATE OPERATORS

```
SELECT [COUNT, SUM, AVG, MAX, MIN(attribute)]
FROM from_list
WHERE COUNT(X)
```

E) ANY, ALL

```
SELECT [DISTINCT] a
FROM from_list
WHERE attribute < != > ALL/ANY (
        SELECT attribute
        FROM X)
```

Find the oldest employee

```
SELECT *
FROM Employee e
WHERE e.age = (SELECT MAX(age) FROM EMPLOYEE);
```

OR:

```
SELECT *
FROM EMPLOYEE
WHERE E.age > ALL
         (SELECT E2.age FROM
        EMPLOYEE E2
        WHERE E2.ssn!=E.ssn);
```

*** ALL => ALL in the set**
*** ANY => At least 1 in the set**
ANY HERE WOULD PRODUCE: Find employees who's age is bigger than AT least somebody's else age.

F) GROUP BY and HAVING CLAUSE

```
SELECT [DISTINCT] a, b, c…z, SUM(A),
FROM from_list
WHERE qualification
GROUP BY a, b, c…z,
HAVING qualification_on_grouping
```

Example: Find how many sailors belong to each group that has more than 30 members

Sailors(sid, name, rating, age, group)

```
SELECT group, count(*) as c
FROM Sailors
GROUP BY group, c
HAVING c>30;
```

Everything that appears in GROUP BY is also part of the select clause

Query: Find the age of the youngest sailor who is eligible to vote (older than 18 years) for each group with at least 2 such sailors.

```
SELECT group, MIN(age)
FROM Sailor
WHERE age>18
GROUP BY group
HAVING COUNT(*)>1;
```

G) NULLs

unknown or *inapplicable*.
Student(ssn, name, age, addressed)
1321, “John”, null
1421, “John”, 15
1521, “John”, 10
1621, “John”, 15

```
SELECT AVG(age)
FROM Student
```

ANSWER: 15+10+15+0 /4 = 10

```
SELECT AVG(age)
FROM Student
WHERE age IS NOT NULL
```

ANSWER: 15 + 10 + 15 /4 = 13.33

Find all student that don’t have their age in the system

```
SELECT *
FROM Student
WHERE AGE IS NULL;
```

H) Nested Queries in the FROM clause (Not implemented in many DBMS systems)

give me a list of salaries (above $20000) where each salary represents the MAX salary of some particular age.

```
SELECT TEMP.salary
FROM (SELECT E.age, MAX(salary) AS salary
      FROM EMPLOYEE
      GROUP BY E.age
      ) AS TEMP
WHERE TEMP.salary>2000;
```