SQL FUNDAMENTALS

31.03.2014
(Muscat, Oman)

OUTLINE

- Definition of SQL
- Main Categories Of Sql
- Basic Sql Commands
- Nested Queries and Subqueries
- Set Operators (minus, in, not in, all, some, intersect, exists)
- Aggregate Functions
- Join Operations
- Using View and Matearilized View
- Sequences
- Triggers And The Purpose Of Triggers

SQL DEFINITION

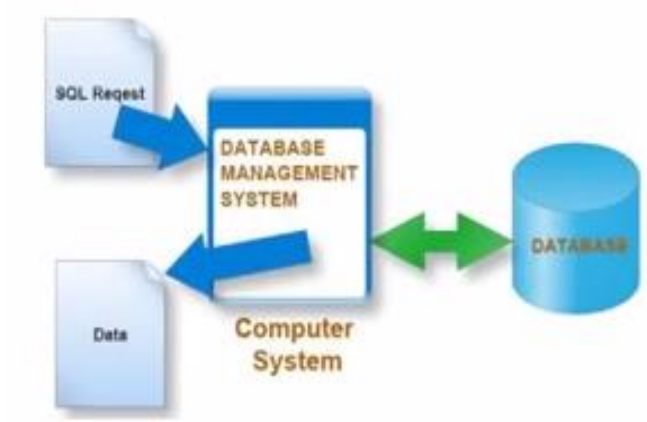
SQL (**Structured Query Language)** is
a special-purpose programming language
designed for managing data held in a RDBMS

First developed in the early 1970s at IBM

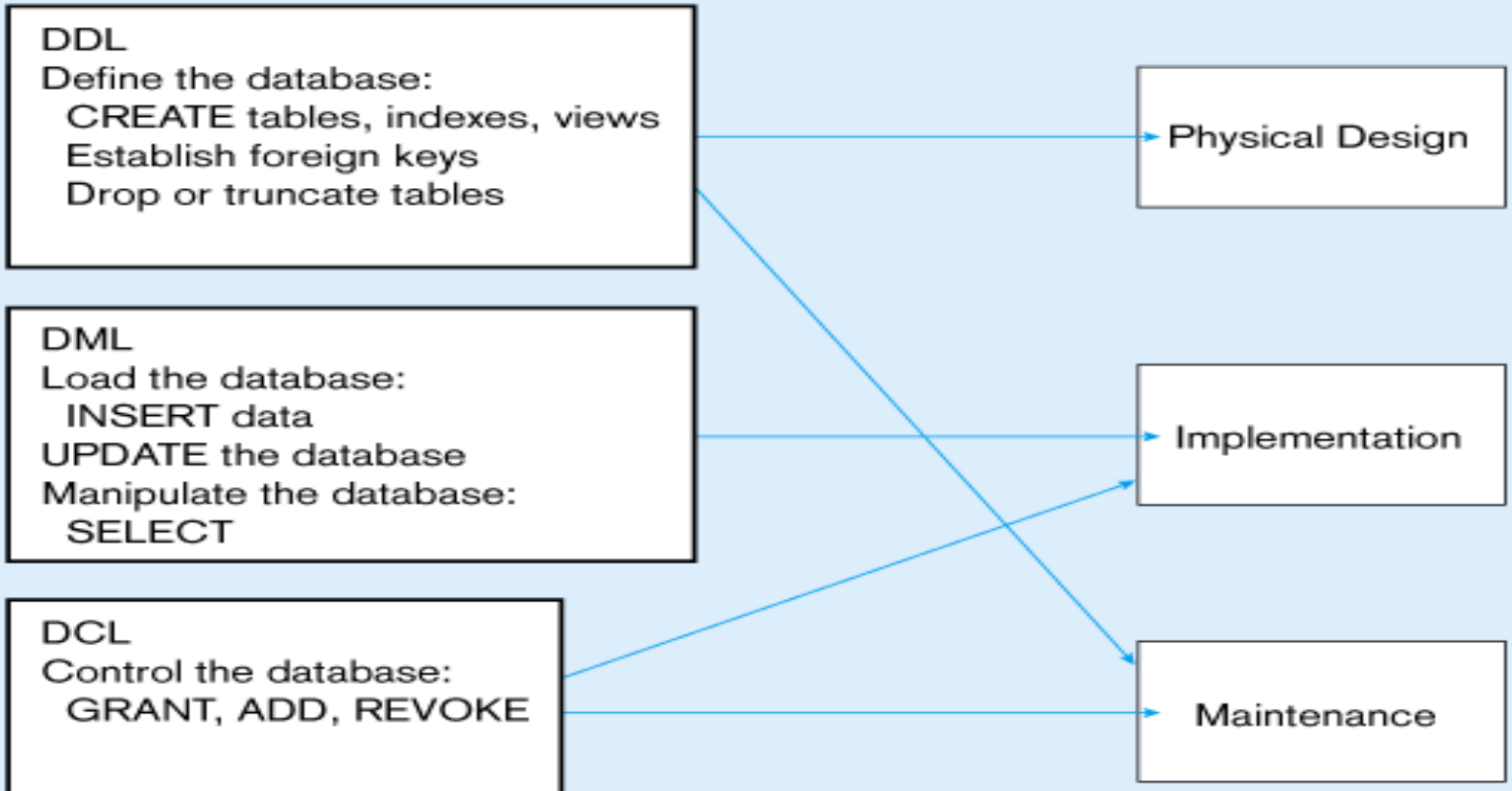
Commercially released by Relational Software Inc.

Became a standard of the

American National Standards Institute (ANSI) in 1986
International Organization for Standards (ISO) in 1987



MAIN CATEGORIES



Sql Commands Overview

CREATE TABLE <name> (<field> <domain>, …)

INSERT INTO <name> (<field names>)
VALUES (<field values>)

DELETE FROM <name>
WHERE <condition>

UPDATE <name>
SET <field name> = <value>
WHERE <condition>

SELECT <fields> (distinct is usable here)
FROM <name>
WHERE <condition>

Some Examples

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	10.10.1998
22	102	10.10.1998
22	103	10.08.1998
22	104	10.07.1998
31	102	11.10.1998
31	103	11.06.1998
31	104	11.12.1998
64	101	09.05.1998
64	102	09.08.1998
74	103	09.08.1998

Boats

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

*Live Demo : install and use a DBMS for SQL s

Examples of Basic SQL Queries

Example : select **distinct** sname, age from sailors

Q1 : Find all sailors with a rating above 7.

Q2 : Find the sailors that rating>7

Q3 : Find the sailors that whose name starts with ‘B’ letter

Q4 : Find the names of the sailors who reserved at least one boat

Nested Queries and Subqueries

A subquery is a select-from-where expression
that is nested within another query.

Example Query:

Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

Using Set Operators (MINUS, NOT IN, NOT EXISTS, IN, EXISTS, INTERSECT)

Syntax : Query1 <Set Operator> Query2

Example1: (difference)

Select * from staff_1 **Minus** Select * from staff2;

Select * from staff_1 where staff_number **not in** (select staff_number from staff_2);

Select * from staff_1 a where **not exists** (select * staff_2 b where a.sid=b.sid);

Example2: (intersect)

Select * from staff_1 **intersect** Select * from staff2;

Select * from staff_1 where staff_number **in** (select staff_number from staff_2);

Select * from staff_1 a where **exists** (select * staff_2 b where a.sid=b.sid);

Exercises:

Q5: Find the sailors that rating of the one equals 3 times of the other
(hint: result is Art, Bob, Horatio)

Q6:Find the names of sailors who have red or green boat

Q7: Find the names of sailors who have reserved a red and a green boat

Q8: Find the sname of sailors who have reserved red boats but not green boats

*Live Demo

Answers:

Q7 Answer -1 (It is a complex query)

```
Select  s.sname
from sailors  s, reserves r 1, boats b1, r 2, boats b2
where  s.sid = r1.sid  and r1.bid=b1.bid
            and s.sid = r2.sid  and r2.bid=b2.bid
            and  b1.color='red'  and  b2.color='green';
```

Q7 Answer -2 – Using Intersect operation (easily to understand, write)

```
Select  s.sname
from sailors  s, reserves r , boats b
where  s.sid = r.sid  and r.bid=b.bid and b.color='red'
INTERSECT
Select  s2.sname
from sailors  s2, reserves r 2, boats b2
where  s2.sid = r2.sid  and r2.bid=b2.bid and b2.color='red' ;
```

Q8 Answer:
Find the sname of sailors who have reserved red boats but not green boats.

```
Select  s.sname
from sailors  s, reserves r , boats b
where  s.sid = r.sid  and r.bid=b.bid and b.color='red'
MINUS
Select  s2.sname
from sailors  s2, reserves r2, boats b2
where  s2.sid = r2.sid  and r2.bid=b2.bid and b2.color='green';
```

ALL, SOME,ANY

They are used to compare two datasets

Example:

Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

Answer-1 :

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

Same query using **some** clause

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```

Same query using **any** clause

```
select name
from instructor
where salary > any (select salary
                    from instructor
                    where dept_name = 'Biology');
```

Example Query for All Clause

Find the names of all instructors whose salary is greater than the salary of **all** instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

UNION vs UNION ALL

They concetane the result of two different SQLs.

They differ in the way they handle **duplicates**.

Selected columns need to be of the same data type

There is a performance hit when using UNION vs UNION ALL,

AGGREGATE FUNCTIONS

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Find the average salary of instructors in the Computer Science department :

select avg (*salary*) f**rom** *instructor* **where** *dept_name*= ’Comp. Sci.’;

Find the total number of instructors who teach a course in the Spring 2010 semester:

select count (**distinct** *ID*) **from** *teaches* **where** *semester* = ’Spring’ **and** *year* = 2010;

..GROUP BY

Attributes in select clause outside of aggregate functions must appear in group by list

/* erroneous query */

```
select city, student_name, count(*), avg(average_note) note from student
group by city
```

Used for queries on single or multiple tables
Clauses of the SELECT statement:

SELECT

List the **columns** (and expressions) that should be returned from the query

FROM

Indicate the **table**(s) or view(s) from which data will be obtained

WHERE

Indicate the **conditions** under which a **row** will be included in the result

GROUP BY

Indicate **columns** to group the results

HAVING

Indicate the **conditions** under which a **group** will be included

ORDER BY

Sorts the result according to specified **columns**

AGGREGATE FUNCTIONS EXERCISES

*Live Demo :

Exercise:

Using STUDENT table

STUDENT(student_name, lesson_name, note)

Find:

a) The count of lessons for each student

b) The average note, and maximum note of each student

c) Find the students that have more than 1 record for the same lesson

Solutions of the Exercise:

```
select student_name,  max(note), avg(note), count(lesson_name) from student
group by student_name;

select student_name, lesson_name, count(*) from student
group by student_name, lesson_name
having count(*) > 1;
```

JOIN OPERATIONS

Join operations take two relations and return as a result another relation.

A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition).
It also specifies the attributes that are present in the result of the join

The join operations are typically used as subquery expressions in the **from clause**

JOIN EXAMPLES

Course :

COURSE_ID	TITLE	DEPT_ID	CREDITS
BIO-301	Genetics	1	4
CS-190	Game Design	2	4
CS-315	Robotics	2	3
MT-101	Mathematics	4	3

Department:

DEPT_ID	DEPT_NAME
1	Biology
2	Computer Science
3	Statistics

JOIN EXAMPLES (Cont'd)

```
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1, department t2
where t1.dept_id = t2.dept_id;
```

COURSE_ID	TITLE	CREDITS	DEPT_ID	DEPT_NAME
BIO-301	Genetics	4	1	Biology
CS-190	Game Design	4	2	Computer Science
CS-315	Robotics	3	2	Computer Science

JOIN EXAMPLES using Right Outer Join

```
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1, department t2
where t1.dept_id(+) = t2.dept_id;
```

```
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1 right outer join department t2
on  t1.dept_id = t2.dept_id;
```

COURSE_ID	TITLE	CREDITS	DEPT_ID	DEPT_NAME
BIO-301	Genetics	4	1	Biology
CS-190	Game Design	4	2	Computer Science
CS-315	Robotics	3	2	Computer Science
			3	Statistics

JOIN EXAMPLES using Left Outer Join

```
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1, department t2
where t1.dept_id = t2.dept_id(+);
```

```
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1 left outer join department t2
on  t1.dept_id = t2.dept_id;
```

COURSE_ID	TITLE	CREDITS	DEPT_ID	DEPT_NAME
BIO-301	Genetics	4	1	Biology
CS-315	Robotics	3	2	Computer Science
CS-190	Game Design	4	2	Computer Science
MT-101	Mathematics	3		

JOIN EXAMPLES using Full Outer Join

```
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1
full outer join
department t2
on  t1.dept_id = t2.dept_id;
```

COURSE_ID	TITLE	CREDITS	DEPT_ID	DEPT_NAME
BIO-301	Genetics	4	1	Biology
CS-190	Game Design	4	2	Computer Science
CS-315	Robotics	3	2	Computer Science
MT-101	Mathematics	3		
			3	Statistics

Some Easy but useful SQL statements

Create table t1 as select * from t2;

Select * from t1 where rownum < 100;

Deleting duplicate records example:

for table Student(student_name, lesson_name, note)

```
delete from STUDENT A1 where exists
(Select 'x' from  STUDENT  A2
where A1.STUDENT_NAME = A2.STUDENT_NAME
and   A1.LESSON_NAME = A2.LESSON_NAME
and   A1.NOTE = A2.NOTE
and A1.ROWID > A2.ROWID);
```

Views and Materialized Views

A view is a result set of a query

By using views, showing only relevant data to the users is provided.

Create view statement : **create view <*view_name*> as < *query expression* >**

•Materializing a view: create a physical table containing all the tuples in the result of the query defining the view (refresh is required)

create materialize view <m*view_name*> as < *query expression* >

SEQUENCES

A sequence is a database object that generates numbers in sequential order. Mostly used to generate primary key for a table

Example:

To create the sequence:

CREATE SEQUENCE customer_seq INCREMENT BY 1 START WITH 100

To use the sequence to enter a record into the database:

INSERT INTO customer (cust_num, name, address)
VALUES (customer_seq.NEXTVAL, 'John Doe', '123 Main St.‘);

TRIGGERS

• Trigger is a procedure that is automatically run by DBMS in response to specified changes to the database, and performs desired events

• Trigger has three **parts** : Event, Condition, Action

Trigger Types Are :

DML triggers for **Insert, Update, Delete**
DDL triggers (instead of) for **Create, Alter, Drop, etc..**

Triggers **can't** be defined for **Select** statement

<!> Transactions statements (commit, savepoint,rollback)
can not be used in Triggers

TRIGGER EXAMPLES

```
create or replace trıgger delete_forbidden
before drop on database
begın
        raıse_applıcatıon_error ( -20000,'dropping table is forbidden');
end;
```

```
create or replace trigger trigger_1
before insert on tbl1
referencıng new as new old as old for each row

begin
  :new.referenced_date :=  sysdate;
end tr_1;
```

SUMMARY

- ✓Basic SQL query has a Select, From,Where
- ✓Use Distinct to avoid duplicates at queries
- ✓SQL provides set operations: Union, Intersect, Minus,
- ✓Queries that have subqueries are called Nested Queries.
- ✓In, Exists, Unique, Any, All, Some is used for Nested Queries
- ✓Aggregators operators are Count, Sum, Average, Max,Min
- ✓Group by and Having
- ✓SQL provides using of Sequences and Triggers

Key Points:

- ✓Be careful about recursive triggers !
- ✓Think about check constraints instead of triggers for database consistency.