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 Materialized View
• Sequences
• Triggers And The Purpose Of Triggers
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.

DDL
Define the database:
- CREATE tables, indexes, views
- Establish foreign keys
- Drop or truncate tables

DML
Load the database:
- INSERT data
- UPDATE the database
- Manipulate the database:
  - SELECT

DCL
Control the database:
- GRANT, ADD, REVOKE

Physical Design
Implementation
Maintenance
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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>29</td>
<td>Brutus</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>64</td>
<td>Horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>Zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>Horatio</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

#### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10.10.1998</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10.10.1998</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10.08.1998</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10.07.1998</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11.10.1998</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11.06.1998</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11.12.1998</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>09.05.1998</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>09.08.1998</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>09.08.1998</td>
</tr>
</tbody>
</table>

#### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>

*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 sidew);\n
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 sidew);
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

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

Same query using `any` clause

```sql
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.

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

They concatenate 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:

```sql
select avg(salary) from instructor where dept_name= 'Comp. Sci.';
```

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

```sql
select count(distinct ID) from teaches where semester = 'Spring' and year = 2010;
```
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

\[
\text{STUDENT}(\text{student\_name, lesson\_name, note})
\]

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:

<table>
<thead>
<tr>
<th>COURSE_ID</th>
<th>TITLE</th>
<th>DEPT_ID</th>
<th>CREDITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MT-101</td>
<td>Mathematics</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Department:

<table>
<thead>
<tr>
<th>DEPT_ID</th>
<th>DEPT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Biology</td>
</tr>
<tr>
<td>2</td>
<td>Computer Science</td>
</tr>
<tr>
<td>3</td>
<td>Statistics</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>COURSE_ID</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>DEPT_ID</th>
<th>DEPT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIO-301</td>
<td>Genetics</td>
<td>4</td>
<td>1</td>
<td>Biology</td>
</tr>
<tr>
<td>CS-190</td>
<td>Game Design</td>
<td>4</td>
<td>2</td>
<td>Computer Science</td>
</tr>
<tr>
<td>CS-315</td>
<td>Robotics</td>
<td>3</td>
<td>2</td>
<td>Computer Science</td>
</tr>
</tbody>
</table>
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;
```
JOIN EXAMPLES using Left Outer Join

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

```sql
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1 left outer join department t2
on t1.dept_id = t2.dept_id;
```
JOIN EXAMPLES using Full Outer Join

```sql
select t1.course_id, t1.title, t1.credits, t2.*
from COURSE t1
full outer join
department t2
on  t1.dept_id = t2.dept_id;
```
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 <mview_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 trigger delete_forbidden
before drop on database
begin
    raise_application_error ( -20000,'dropping table is forbidden');
end;

create or replace trigger trigger_1
before insert on tbl1
referencing 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.