SQL – Structured Query Language

More Details

Example Tables

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	30
.....						
7698	BLAKE	MANAGER		01-MAY-81	3850	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	10

EMPNO: number(4)

ENAME: varchar2(30)

JOB: char(10)

MGR: number(4)

HIREDATE: date

SAL: number(7,2)

DEPTNO: number(2)

Example Tables

DEPT

DEPTNO	DNAME	LOC
10	STORE	CHICAGO
20	RESEARCH	DALLAS
30	SALES	NEW YORK
40	MARKETING	BOSTON

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Sub Query

- Up to now we have only concentrated on simple comparison conditions in a where clause.
  - we have compared a column with a constant
  - we have compared two columns.
- We have already seen for the insert statement, queries can be used for assignments to columns.
- A query result can also be used in a condition of a where clause.
- In such a case the query is called a ***sub query*** and the complete select statement is called a ***nested query***.

Sub Query

- List the name and salary of employees of the department 20 who are leading a project that started before December 31, 1990:
  - ***select ENAME, SAL from EMP where DEPTNO =20 and EMPNO in (select PMGR from PROJECT where PSTART < '31-DEC-90')***
- The sub query retrieves the set of those employees who manage a project that started before December 31, 1990.
- If the employee working in department 20 is contained in this set (in operator), this tuple belongs to the query result set.

Sub Query

- List all employees who are working in a department located in BOSTON:
  - ***select * from EMP where DEPTNO in (select DEPTNO from DEPT where LOC = 'BOSTON')***
- The sub query retrieves only one value (the number of the department located in Boston). So it is possible to use = instead of in.
- As long as the result of a sub query is not known in advance, (whether it is a single value or a set), it is advisable to use the in operator.

Sub Query

- A sub query may use again a sub query in its where clause. Thus conditions can be nested arbitrarily.
- An important class of sub queries are those that refer to its surrounding (sub)query and the tables listed in the from clause, respectively.
- List all those employees who are working in the same department as their manager
  - ***select * from EMP E1 where DEPTNO in (select DEPTNO from EMP [E] where [E.]EMPNO = E1.MGR)***
- What we can see here ? Table can also be given alias in the query.

Sub Query

- ***select * from EMP E1 where DEPTNO in (select DEPTNO from EMP [E] where [E.]EMPNO = E1.MGR)***
- One can think of the evaluation of this query as follows:
  - For each tuple in the table E1, the sub query is evaluated individually.
  - If the condition (where DEPTNO in . . .) evaluates to true, this tuple is selected.
- Note that an alias for the table EMP in the sub query is not necessary since columns without a preceding alias listed there always refer to the innermost query and tables.

Any/All

- For the clause **any**, the condition evaluates to true if there exists at least on row selected by the sub query for which the comparison holds.
- If the sub query yields an empty result set, the condition is not satisfied.
- Retrieve all employees who are working in department 10 and who earn at least as much as any (at least one) employee working in department 30.
  - ***select * from EMP where DEPTNO = 10 and SAL >= any (select SAL from EMP where DEPTNO = 30)***

Any/All

- For the clause **all**, the condition evaluates to true if for all rows selected by the sub query the comparison holds.
- In this case the condition evaluates to true if the sub query does not yield any row or value.
- List all employees who are not working in department 30 and who earn more than all employees working in department 30:
  - ***select * from EMP where DEPTNO <> 30 and SAL > all (select SAL from EMP where DEPTNO = 30)***
- Find the name of the employee with maximum salary.
  - ***select ENAME from EMP where SAL >=all(select SAL from EMP)***

Union/Intersection/Minus

- Sometimes it is useful to combine query results from two or more queries into a single result. SQL supports three set operators which have the pattern:

  <query 1> <set operator> <query 2>

- ***union [all]*** returns a table consisting of all rows either appearing in the result of <query1> or in the result of <query 2>. Duplicates are automatically eliminated unless the clause all is used.
- ***intersect*** returns all rows that appear in both results <query 1> and <query 2>.
- ***minus*** returns those rows that appear in the result of <query 1> but not in the result of <query 2>.

Union/Intersection/Minus

- Assuming we have a table EMP2 that has the same structure and columns as the table EMP.
- All employee numbers and names from both tables.
  - ***select EMPNO, ENAME from EMP union select EMPNO, ENAME from EMP2***
- Employees who are listed in both EMP and EMP2.
  - ***select * from EMP intersect select * from EMP2***
- Employees who are only listed in EMP.
  - ***select * from EMP minus select * from EMP2***
- Each operator requires that both tables have the same data types for the columns to which the operator is applied.

Join

- Joins are very important for relational databases.
- Mainly used when we need to find information that distributes over multiple tables.
- For each salesman, we now want to retrieve the name as well as the number and the name of the department where he is working.
- ENAME – {EMP}, DNAME- {DEPT}, DEPTNO – {EMP, DEPT}

  – ***select ENAME, E.DEPTNO, DNAME from EMP E, DEPT D where E.DEPTNO = D.DEPTNO and JOB = 'SALESMAN'***

Join

- The computation of the query result occurs in the following manner
  - Each row from the table EMP is combined with each row from the table DEPT (this operation is called ***Cartesian Product***). If EMP contains m rows and DEPT contains n rows, we thus get n * m rows.
  - From these rows those that have the same department number are selected (where E.DEPTNO = D.DEPTNO).
  - From this result finally all rows are selected for which the condition JOB = ’SALESMAN’ holds.

Join

- Any number of tables can be combined in a select statement.
- For each project, retrieve its name, the name of its manager, and the name of the department where the manager is working.
  - ***select ENAME, DNAME, PNAME from EMP E, DEPT D, PROJECT P where E.EMPNO = P.MGR and D.DEPTNO = E.DEPTNO***
- It is even possible to join a table with itself:
- List the names of all employees together with the name of their manager.
  - ***select E1.ENAME, E2.ENAME from EMP E1, EMP E2 where E1.MGR = E2.EMPNO***

Join

- For each department, number and the name of the department and also the employees name working in the department.
  - ***select DNAME,EMP.DEPTNO,ENAME from DEPT, EMP where DEPT.DEPTNO = EMP.DEPTNO***
- This is also known as **inner join**
  - ***select DNAME,EMP.DEPTNO,ENAME from DEPT inner join EMP on DEPT.DEPTNO = EMP.DEPTNO***
- Another way is to by **natural join**
  - ***select DNAME,DEPTNO,ENAME from DEPT natural join EMP***

Join

- Another type of join is as **left outer join**. Here all the rows from left table will be included in the result.
  - ***select DNAME,EMP.DEPTNO,ENAME from DEPT left outer join EMP on DEPT.DEPTNO = EMP.DEPTNO***
  - ***select DNAME,EMP.DEPTNO,ENAME from DEPT,EMP where DEPT.DEPTNO = EMP.DEPTNO(+)***
- Another type of join is as **right outer join**. Here all the rows from right table will be included in the result.
  - ***select DNAME,EMP.DEPTNO,ENAME from DEPT right outer join EMP on DEPT.DEPTNO = EMP.DEPTNO***
  - ***select DNAME,EMP.DEPTNO,ENAME from DEPT,EMP where DEPT.DEPTNO(+) = EMP.DEPTNO***

Join

- Another type of join is as **full outer join**. Here all the rows from left table and right table will be included.
  - ***select DNAME,EMP.DEPTNO,ENAME from DEPT full outer join EMP on DEPT.DEPTNO = EMP.DEPTNO***
- In all outer joins some of the resulting columns may be null.

Exists/Not Exists

- exists (sub-select)
  - The predicate is true if the sub-select results in a nonempty set of values. The predicate is false otherwise.
- not exists (sub-select)
  - The predicate is true if the sub-select results in an empty set of values. The predicate is false otherwise.
- Find the employee name who is leading at least one project.
  - ***select ENAME from EMP where exists (select 'a' from PROJECT where PMGR=EMP.EMPNO)***
- Find the employee name who is leading no project.
  - ***select ENAME from EMP where not exists (select 'a' from PROJECT where PMGR=EMP.EMPNO)***

Exists/Not Exists

- Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name from student S
    where not exists (
    (
     select course_id from course where dept_name='Biology'
     )
    minus
    (
    select T.course_id from takes T where S.ID = T.ID
    )
 )
```

View

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know a employees name, job and dept but has no need to see the salary.
- This person should see a relation described, in SQL, by
  - ***select ENAME, JOB, DEPTNO from EMP***
- A view provides a mechanism to hide certain data from the view of certain users.

View

- In Oracle the SQL command to create a view has the form

```
create [or replace] view <view-name> [(<column(s)>)] as
    <select-statement> [with check option [constraint <name>]];
```

- The optional clause or replace re-creates the view if it already exists. If <column(s)> is not specified in the view definition, the columns of the view get the same names as the attributes listed in the select statement.
- When a view is created, the query is stored in the database, the expression is substituted into queries using the view.

View

- The following view contains the name, job title and the annual salary of employees working in the department 20.
  - ***create view DEPT20 as select ENAME, JOB, SAL*12 as ANNUAL SALARY from EMP where DEPTNO = 20***
- In the select statement the column alias ANNUAL SALARY is specified for the expression SAL*12 and this alias is taken by the view.
- An alternative formulation of the above view definition is
  - ***create view DEPT20 (ENAME, JOB, ANNUAL SALARY) as select ENAME, JOB, SAL * 12 from EMP where DEPTNO = 20***

View

- A view can be used in the same way as a table, that is, rows can be retrieved from a view or rows can even be modified.
- In Oracle SQL no insert, update, or delete modifications on views are allowed that use one of the following constructs in the view definition:
  - joins
  - aggregate function such as sum, min, max etc
  - set-valued sub queries (in, any, all) or test for existence (exists)
  - group by clause or distinct clause
  - new/modified row does not meet the view definition

Sequence

- Sequence are used for automatic number generation.
- How to create a sequence
  - ***create sequence mysequence minvalue 1 start with 1 increment by 1***
- How to use any sequence
  - ***insert into DEPT (select mysequence.nextval,'NAME','LOC')***
- How to modify
  - ***Alter sequence mysequence increment by 5***

Alter

- It is possible to modify the structure of a table (the relation schema) even if rows have already been inserted into this table.
- To add a column to an existing table:
  - ***alter table table_name add column_name column_definition***
  - For example:
  - ***alter table EMP add DOB date***

Alter

- To add multiple columns to an existing table:
  - ***alter table table_name add (***
    column_1 column_definition,
    column_2 column_definition,
    column_n column_definition)
- To modify a column in an existing table:
  - ***alter table table_name modify column_name column_type***
  - For example:
  - ***alter table EMP modify ENAME varchar2(100) not null***

Alter

- To modify multiple columns in an existing table:
  - ***alter table table_name modify (***
    column_1 column_type,
    column_2 column_type,
    column_n column_type)
- To drop a column in an existing table:
  - ***alter table table_name drop column column_name***
  - For example:
  - ***alter table EMP drop column DOB***

Alter

- To rename a column in an existing table:
  - ***alter table table_name rename column old_name to new_name***
- For example:
  - ***alter table EMP rename column SAL to SALARY***
- To rename a table:
  - ***alter table table_name rename to new_table_name***
- For example:
  - ***alter table EMP rename to EMPNEW***

Alter

- It is also possible to add/drop constraints using the alter command.
- To add a table-level constraint to an existing table.
  - ***alter table table_name add constraint constraint_name constraint_type (column_name)***
  - ***alter table EMP add constraint EMP_PK primary key (EMPNO)***
- To drop a constraint on an existing table using constraint name
  - ***alter table table_name drop constraint constraint_name***
  - ***alter table EMP drop constraint EMP_PK***