SQL interview questions and answers

What is SQL and where does it come from?

Structured Query Language (SQL) is a language that provides an interface to relational database systems. SQL was developed by IBM in the 1970s for use in System R, and is a de facto standard, as well as an ISO and ANSI standard. SQL is often pronounced SEQUEL.

In common usage SQL also encompasses DML (Data Manipulation Language), for INSERTs, UPDATEs, DELETEs and DDL (Data Definition Language), used for creating and modifying tables and other database structures.

The development of SQL is governed by standards. A major revision to the SQL standard was completed in 1992, called SQL2. SQL3 support object extensions and are (partially?) implemented in Oracle8 and 9.

What are the difference between DDL, DML and DCL commands?

DDL is Data Definition Language statements. Some examples:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command

DML is Data Manipulation Language statements. Some examples:

- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency

DCL is Data Control Language statements. Some examples:

- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT
- SET TRANSACTION - Change transaction options like what rollback segment to use

How does one escape special characters when building SQL queries?

The LIKE keyword allows for string searches. The '_' wild card character is used to match exactly one character, '%' is used to match zero or more occurrences of any characters. These characters can be escaped in SQL. Example:

```
SELECT name FROM emp WHERE id LIKE '%\_%' ESCAPE '\';
```

Use two quotes for every one displayed. Example:

```
SELECT 'Franks''s Oracle site' FROM DUAL;
SELECT 'A ''quoted'' word.' FROM DUAL;
SELECT 'A ''''double quoted'''' word.' FROM DUAL;
```

How does one eliminate duplicates rows from a table?

Choose one of the following queries to identify or remove duplicate rows from a table leaving only unique records in the table:

Method 1:

```
SQL> DELETE FROM table_name A WHERE ROWID > (
  2    SELECT min(rowid) FROM table_name B
  3    WHERE A.key_values = B.key_values);
```

Method 2:

```
SQL> create table table_name2 as select distinct * from table_name1;
SQL> drop table_name1;
SQL> rename table_name2 to table_name1;
SQL> -- Remember to recreate all indexes, constraints, triggers, etc
on table...
```

Method 3: (thanks to Dennis Gurnick)

```
SQL> delete from my_table t1
SQL> where  exists (select 'x' from my_table t2
SQL>                where t2.key_value1 = t1.key_value1
SQL>                  and t2.key_value2 = t1.key_value2
SQL>                  and t2.rowid      > t1.rowid);
```

Note: One can eliminate N^2 unnecessary operations by creating an index on the joined fields in the inner loop (no need to loop through the entire table on each pass by a record). This will speed-up the deletion process.

Note 2: If you are comparing NOT-NULL columns, use the NVL function. Remember that NULL is not equal to NULL. This should not be a problem as all key columns should be NOT NULL by definition.

How does one generate primary key values for a table?

Create your table with a NOT NULL column (say SEQNO). This column can now be populated with unique values:

```
SQL> UPDATE table_name SET seqno = ROWNUM;
```

or use a sequences generator:

```
SQL> CREATE SEQUENCE sequence_name START WITH 1 INCREMENT BY 1;
SQL> UPDATE table_name SET seqno = sequence_name.NEXTVAL;
```

Finally, create a unique index on this column.

How does one get the time difference between two date columns?

Look at this example query:

```
select floor(((date1-date2)*24*60*60)/3600)
       || ' HOURS ' ||
       floor((((date1-date2)*24*60*60) -
       floor(((date1-date2)*24*60*60)/3600)*3600)/60)
       || ' MINUTES ' ||
       round((((date1-date2)*24*60*60) -
       floor(((date1-date2)*24*60*60)/3600)*3600 -
       (floor((((date1-date2)*24*60*60) -
       floor(((date1-date2)*24*60*60)/3600)*3600)/60)*60)))
       || ' SECS ' time_difference
from   ...
```

If you don't want to go through the floor and ceiling math, try this method (contributed by Erik Wile):

```
select to_char(to_date('00:00:00','HH24:MI:SS') +
            (date1 - date2), 'HH24:MI:SS') time_difference
from ...
```

Note that this query only uses the time portion of the date and ignores the date itself. It will thus never return a value bigger than 23:59:59.

How does one add a day/hour/minute/second to a date value?

The SYSDATE pseudo-column shows the current system date and time. Adding 1 to SYSDATE will advance the date by 1 day. Use fractions to add hours, minutes or seconds to the date. Look at these examples:

```
SQL> select sysdate, sysdate+1/24, sysdate +1/1440, sysdate +
1/86400 from dual;

SYSDATE              SYSDATE+1/24         SYSDATE+1/1440
SYSDATE+1/86400
-------------------- -------------------- --------------------
--------------------
03-Jul-2002 08:32:12 03-Jul-2002 09:32:12 03-Jul-2002 08:33:12
03-Jul-2002 08:32:13
```

The following format is frequently used with Oracle Replication:

```
select sysdate NOW, sysdate+30/(24*60*60) NOW_PLUS_30_SECS from
dual;

NOW                  NOW_PLUS_30_SECS
-------------------- --------------------
03-JUL-2002 16:47:23 03-JUL-2002 16:47:53
```

How does one count different data values in a column?

Use this simple query to count the number of data values in a column:

```
select my_table_column, count(*)
from   my_table
group  by my_table_column;
```

A more sophisticated example...

```
select dept, sum(  decode(sex,'M',1,0)) MALE,
             sum(  decode(sex,'F',1,0)) FEMALE,
             count(decode(sex,'M',1,'F',1)) TOTAL
from   my_emp_table
group  by dept;
```

How does one count/sum RANGES of data values in a column?

A value x will be between values y and z if GREATEST(x, y) = LEAST(x, z). Look at this example:

```
select f2,
       sum(decode(greatest(f1,59), least(f1,100), 1, 0)) "Range
60-100",
       sum(decode(greatest(f1,30), least(f1, 59), 1, 0)) "Range
30-59",
       sum(decode(greatest(f1, 0), least(f1, 29), 1, 0)) "Range
00-29"
from   my_table
group  by f2;
```

For equal size ranges it might be easier to calculate it with DECODE(TRUNC(value/range), 0, rate_0, 1, rate_1, ...). Eg.

```
select ename "Name", sal "Salary",
       decode( trunc(f2/1000, 0), 0, 0.0,
                                  1, 0.1,
                                  2, 0.2,
                                  3, 0.31) "Tax rate"
from   my_table;
```

Can one retrieve only the Nth row from a table?

Shaik Khaleel provided this solution to select the Nth row from a table:

```
SELECT * FROM (
   SELECT ENAME,ROWNUM RN FROM EMP WHERE ROWNUM < 101 )
WHERE  RN = 100;
```

Note: Note: In this first it select only one more than the required row, then it selects the required one. Its far better than using MINUS operation.

Ravi Pachalla provided this solution:

```
SELECT f1 FROM t1
WHERE  rowid = (
   SELECT rowid FROM t1
```

```
            WHERE  rownum <= 10
           MINUS
            SELECT rowid FROM t1
            WHERE  rownum < 10);
```

Alternatively...

```
SELECT * FROM emp WHERE rownum=1 AND rowid NOT IN
   (SELECT rowid FROM emp WHERE rownum < 10);
```

Please note, there is no explicit row order in a relational database. However, this query is quite fun and may even help in the odd situation.

Can one retrieve only rows X to Y from a table?

Shaik Khaleel provided this solution to the problem:

```
SELECT * FROM (
    SELECT ENAME,ROWNUM RN FROM EMP WHERE ROWNUM < 101
) WHERE  RN between 91 and 100 ;
```

Note: the 101 is just one greater than the maximum row of the required rows (means x= 90, y=100, so the inner values is y+1).

Another solution is to use the MINUS operation. For example, to display rows 5 to 7, construct a query like this:

```
SELECT *
FROM   tableX
WHERE  rowid in (
   SELECT rowid FROM tableX
   WHERE rownum <= 7
  MINUS
   SELECT rowid FROM tableX
   WHERE rownum < 5);
```

Please note, there is no explicit row order in a relational database. However, this query is quite fun and may even help in the odd situation.

How does one select EVERY Nth row from a table?

One can easily select all even, odd, or Nth rows from a table using SQL queries like this:

Method 1: Using a subquery

```
SELECT *
FROM   emp
WHERE  (ROWID,0) IN (SELECT ROWID, MOD(ROWNUM,4)
                     FROM   emp);
```

Method 2: Use dynamic views (available from Oracle7.2):

```
SELECT *
FROM   ( SELECT rownum rn, empno, ename
         FROM emp
       ) temp
WHERE  MOD(temp.ROWNUM,4) = 0;
```

Please note, there is no explicit row order in a relational database. However, these queries are quite fun and may even help in the odd situation.

How does one select the TOP N rows from a table?

Form Oracle8i one can have an inner-query with an ORDER BY clause. Look at this example:

```
SELECT *
FROM   (SELECT * FROM my_table ORDER BY col_name_1 DESC)
WHERE  ROWNUM < 10;
```

Use this workaround with prior releases:

```
SELECT *
  FROM my_table a
 WHERE 10 >= (SELECT COUNT(DISTINCT maxcol)
                FROM my_table b
               WHERE b.maxcol >= a.maxcol)
 ORDER BY maxcol DESC;
```

How does one code a tree-structured query?

Tree-structured queries are definitely non-relational (enough to kill Codd and make him roll in his grave). Also, this feature is not often found in other database offerings.

The SCOTT/TIGER database schema contains a table EMP with a self-referencing relation (EMPNO and MGR columns). This table is perfect for tesing and demonstrating tree-structured queries as the MGR column contains the employee number of the "current" employee's boss.

The LEVEL pseudo-column is an indication of how deep in the tree one is. Oracle can handle queries with a depth of up to 255 levels. Look at this example:

```
select  LEVEL, EMPNO, ENAME, MGR
  from  EMP
connect by prior EMPNO = MGR
  start with MGR is NULL;
```

One can produce an indented report by using the level number to substring or lpad() a series of spaces, and concatenate that to the string. Look at this example:

```
select lpad(' ', LEVEL * 2) || ENAME ........
```

One uses the "start with" clause to specify the start of the tree. More than one record can match the starting condition. One disadvantage of having a "connect by prior" clause is that you cannot perform a join to other tables. The "connect by prior" clause is rarely implemented in the other database offerings. Trying to do this programmatically is difficult as one has to do the top level query first, then, for each of the records open a cursor to look for child nodes.

One way of working around this is to use PL/SQL, open the driving cursor with the "connect by prior" statement, and the select matching records from other tables on a row-by-row basis, inserting the results into a temporary table for later retrieval.

How does one code a matrix report in SQL?

Look at this example query with sample output:

```
SELECT  *
FROM  (SELECT job,
              sum(decode(deptno,10,sal)) DEPT10,
              sum(decode(deptno,20,sal)) DEPT20,
```

```
                sum(decode(deptno,30,sal)) DEPT30,
                sum(decode(deptno,40,sal)) DEPT40
           FROM scott.emp
          GROUP BY job)
ORDER BY 1;

JOB           DEPT10     DEPT20     DEPT30     DEPT40
--------- ---------- ---------- ---------- ----------
ANALYST                    6000
CLERK           1300       1900        950
MANAGER         2450       2975       2850
PRESIDENT       5000
SALESMAN                                5600
```

How does one implement IF-THEN-ELSE in a select statement?

The Oracle **decode** function acts like a procedural statement inside an SQL statement to return different values or columns based on the values of other columns in the select statement.

Some examples:

```
select decode(sex, 'M', 'Male', 'F', 'Female', 'Unknown')
from   employees;

select a, b, decode( abs(a-b), a-b, 'a > b',
                               0,   'a = b',
                                    'a < b') from  tableX;

select decode( GREATEST(A,B), A, 'A is greater OR EQUAL than
B', 'B is greater than A')...

select decode( GREATEST(A,B),
          A, decode(A, B, 'A NOT GREATER THAN B', 'A GREATER
THAN B'),
           'A NOT GREATER THAN B')...
```

Note: The decode function is not ANSI SQL and is rarely implemented in other RDBMS offerings. It is one of the good things about Oracle, but use it sparingly if portability is required.

From Oracle 8i one can also use CASE statements in SQL. Look at this example:

```
SELECT ename, CASE WHEN sal>1000 THEN 'Over paid' ELSE 'Under
paid' END
FROM   emp;
```

How can one dump/ examine the exact content of a database column?

```
SELECT DUMP(col1)
FROM tab1
WHERE cond1 = val1;

DUMP(COL1)
------------------------------------
```

```
        Typ=96 Len=4: 65,66,67,32
```

For this example the type is 96, indicating CHAR, and the last byte in the column is 32, which is the ASCII code for a space. This tells us that this column is blank-padded.

Can one drop a column from a table?

From Oracle8i one can DROP a column from a table. Look at this sample script, demonstrating the *ALTER TABLE table_name DROP COLUMN column_name;* command.

Other workarounds:

```
1. SQL> update t1 set column_to_drop = NULL;
   SQL> rename t1 to t1_base;
   SQL> create view t1 as select <specific columns> from t1_base;

2. SQL> create table t2 as select <specific columns> from t1;
   SQL> drop table t1;
   SQL> rename t2 to t1;
```

Can one rename a column in a table?

No, this is listed as Enhancement Request 163519. Some workarounds:

```
1. -- Use a view with correct column names...
   rename t1 to t1_base;
   create view t1 <column list with new name> as select * from t1_base;

2. -- Recreate the table with correct column names...
   create table t2 <column list with new name> as select * from t1;
   drop table t1;
   rename t2 to t1;

3. -- Add a column with a new name and drop an old column...
   alter table t1 add ( newcolame datatype );
   update t1 set newcolname=oldcolname;
   alter table t1 drop column oldcolname;
```

-

How can I change my Oracle password?

Issue the following SQL command: `ALTER USER <username> IDENTIFIED BY <new_password>` `/`

From Oracle8 you can just type "password" from SQL*Plus, or if you need to change another user's password, type "password user_name".

How does one find the next value of a sequence?

Perform an "ALTER SEQUENCE ... NOCACHE" to unload the unused cached sequence numbers from the Oracle library cache. This way, no cached numbers will be lost. If you then select from the USER_SEQUENCES dictionary view, you will see the correct high water mark value that would be returned for the next NEXTVALL call. Afterwards, perform an "ALTER SEQUENCE ... CACHE" to restore caching.

You can use the above technique to prevent sequence number loss before a *SHUTDOWN ABORT*, or any other operation that would cause gaps in sequence values.

Workaround for snapshots on tables with LONG columns

You can use the SQL*Plus COPY command instead of snapshots if you need to copy LONG and LONG RAW variables from one location to another. Eg:

```
COPY TO SCOTT/TIGER@REMOTE     -
CREATE IMAGE_TABLE USING       -
       SELECT IMAGE_NO, IMAGE  -
       FROM   IMAGES;
```

Note: If you run Oracle8, convert your LONGs to LOBs, as it can be replicated.
