|  |  |  |
| --- | --- | --- |
| 5.4 Alias  Sometimes a column name is a vague indicator of the data that's displayed. To better describe the data displayed in the output, you can substitute a column alias for the column name in query results. You need to keep some guidelines in mind when using a column alias. If the column alias contains spaces or special symbols, or if you don't want it to appear in all uppercase letters, you must enclose it in quotation marks ( ). By default, column headings shown in query results are capitalized. Using quotation marks overrides this default setting. If the column alias consists of only one word without special symbols, it doesn't need to be enclosed in quotation marks  --Can modify the headings with your choice of alias. This does not change the underlying --columnname in the table. It only displays a different heading for this one SQL statement.  SELECT fname, Inane AS lastname FROM patient;  --Can also come up with an alias without using the AS keyword.  SELECT fname, Inane lastname FROM patient;  —If an alias is comprised of multiple words then enclose it in double quotes.  SELECT fname, Iname "Last Name" FROM patient; | 5.5 Concatenation  Use the concatenation operator (l l) to concatenate, or combine, data from columns with string literals.  --SELECT fname I I Inamel I gender FROM patient;  The CONCAT function can also be used to concatenate data from two columns. The main difference between the concatenation operator and the CONCAT function is that you can combine a long list of columns and string literals with the concatenation operator. By contrast, you can combine only two items (columns or string literals) with the CONCAT function.  —Instead of the pipe symbols, the concat function can also be used. It only takes two arguments.  SELECT CONCAT (fname, Iname) FROM patient;  SELECT CONCAT (fname, Inarne I I gender) FROM patient; | 5.6 LTRIWRTRIM/TRIM  You can use the LTRIM function to remove a specific string of characters from the left side of data values. The syntax of the LTRIM function is LTRlM(c, s), where c represents the field to modify, and s represents the string to remove from the left side of data.  Oracle also supports the RTRIM function to remove specific characters from the right side of data values. The syntax of the RTRIM function is RTRlM(c, s). The c represents the field to modify, and s represents the string to remove from the right side of data. The TRIM function has several variations that can potentially replace both RTRIM and LTRIM.  --Thts record is being inserted so that we can remove the hyphens using the trim function.  TUSEPT INTO patient VALUES (777, Doe ' 05-DEC1990 ' , 90000, ' Davis' , ' CA ' ) ;  --Removes the dashes from the left side and right side: Keep in mind this is for display purposes only. It doesn't change the actual data in the table.  SELECT LTRIM(fname, '-' ) , RTPIM(fname, ) FROM patient;  —Can also use the trim function with LEADING or TRAILING options. Without those options it will --100k at both the left and the right sides.  SELECT TRIM (LEADING FROM fname) , TRIM (TRAILING' - FROM fname) , TRIM ( FROM fname) FP.OM patient; |
| 5.7 DISTINCT or UNIQUE  The DISTINCT/UNIQUE keyword eliminates duplicate values in the results.  SELECT city, state FROM patient;  --Gets rid of the duplicate cities.  SELECT DISTINCT city FROM patient;  --Gets rid of the duplicate city, state combination like a composite key.  SELECT DISTINCT city, state FROM patient;  --INVALID. Must apply DISTINCT to the entire row.  SELECT city, DISTINCT state FROM patient;  --Can also use the UNIQUE keyword instead of distinct. SELECT UNIQUE city, state FROM atient; | 5.8 DUAL table  Any of the single- row functions covered in this chapter can be used with the DUAL table  —The reason why we want to use the dual table is because there is only one record. In this  —instance, notice it displays the literal text (hello) fror every record  SELECT 'hello', fname FROM patient; SELECT 'hello' FROM patient;  —Displays hello only one time  SELECT 'hello' FROM dual; | 5.9 INITCAP  The initcap function sets the first character in each word to uppercase and the rest to lowercase. The syntax for the initcap function is: initcap( stringl ) where stringl is the string argument whose first character in each word will be converted to uppercase and all remaining characters converted to lowercase.  --Does not change, Just displays. First letter is upper cased.  SELECT INITCAP( 'tech on the net') FROM dual;  --First letter is uppercased.  SELECT INITCAP 'GEORGE SOROS' ) FROM dual;  SELECT iUITCAP(fnarne) FROM patient; |
| 5.10 SUBSTR, INSTR and REPLACE  The INSTR (instring) function searches a string for a specified set of characters or a sub-string, and then returns a numeric value representing the first character position in which the substring is found. If the substring doesn't exist in the string value, a 0 ( zero) is returned. Two arguments must be provided to the INSTR function: the string value to search and the characters or substring (enclosed in single quotes) to locate. Two optional arguments are also available: start position, indicating on which character of the string value the search should begin, and occurrence, which is the instance of the search value to locate ( that is, first occurrence, second occurrence, and so on). By default, the search begins at the beginning of the string value, and the position of the first occurrence is located.  --Return first occurrence of 'e' .  SELECT INSTR ('Tech on the net', FROM DUAL;  --The first occurrence of  SELECT INSTR ('Tech on the net', e' , 1, 1) FROM DUAL;  --The second occurrence of l e i  SELECT INSTR ('Tech on the net' ,,3,1, 2) FROM DUAL;  --The third occurrence of  SELECT INSTR ('Tech on the net' ,’e’1, 3) FROM DUAL;  --Looks for the first occurrence of the letter (l).  SELECT (fname, ) fname FROM | **Example 5.10b (Substr)**  **You can use the SUBSTR function to return a substring (a portion of a string). The syntax of this function is SUBSTR( c, p, l), where c represents the character string, p represents the beginning character position for the extraction, and I represents the length of the string to return in the query results.**  **Extracts two characters, starting from the 6th position.**  **SELECT SUBSTR ('This is a test' , 6, 2) FROM DUAL;**  **- If the second parameter is omitted, substr will return the entire string.**  **SELECT SUBSTR ('This is a test', 6) FROM DUAL;**  **Extracts four characters, starting from the 1st position.**  **SELECT SUBSTR ( TechOnTheNet'**  **, 1, 4) FROM DUAL;**  **--lf start\_position is a negative number, then it starts from the end of the string and counts backwards.**  **SELECT SUBSTR (**  **SELECT SUBSTR ( t TechOnTheNeE' -3, 3) FROM DUAL; SELECT SUBSTR ('TechOnTheNet', -6, 3) FROM DUAL; -8, 2) FROM DUAL** | **Example 5.10c (Replace)**  **The REPLACE function is similar to the "search and replace" function used in many programs. It looks for the occurrence of a specified string of characters and, if found, substitutes it with another set of characters. The syntax of the REPLACE function is REPLACE( c, s, r), where c represents the field to search, s represents the string of characters to find, and r represents the string of characters to substitute for s**  **-- Gets rid of everything in the second argument. It is looking for the exact pattern. SELF,CT REPI.,ACE, ( ' 123123tech' , ' 123' ) FROM DUAL;**  **-- Doesn't matter where the 123 is.**  **SELECT REPLACE ( ' 123tech123' , ' 123' ) FROM DUAL;**  **—Replaces all every 2 with a 3.**  **SELECT REPLACE ( t 222Lech' , FROM DUAL;**  **-- Gets rid of all the zeros.**  **SELECT FROM DUAL;**  **Replaces the zeros with blank spaces.**  **SELECT REPLACE ( ' 0000123' , ' ) FROM DUAL;** |
| 5.11 LPAD, RPAD  The LPAD function can be used to pad, or fill in, the area to the left of a character string with a specific character— or even a blank space. The syntax of the LPAD function is LPAD( c, l, s), where c represents the character string to pad, I represents the length of the character string after padding, and s represents the symbol or character ( enclosed in single quotes) to use as padding, RPAD is similar except that it pads from the right hand side.  —Pad the left hand side with dots.  SELECT LAD (fname, 20, ' . ' ) FROM patient;  —Pad the right hand side with dots.  SELECT READ (fname, 20, ' t ) FROM patient; | TRUNC, ROUND, FLOOR, CEIL  The syntax of the ROUND function is ROUND (d, u), where d represents the date data, or field, to round, and u represents the unit to use for rounding. A date can be rounded by the unit of month or year. | Exantple 5.12b (Trunc)  At times you need to truncate, rather than round, numeric data. You can use the TRUNC ( truncate) function to truncate a numeric value to a specific position. Any numbers after that position are simply removed (truncated). The syntax of the TRUNC function is TRUNC( n, p), where n represents the numeric data or field to truncate, and p represents the position of the digit where data should be truncated.  SELECT FROM DUAL;  SELECT TRUNC (12 . 5) FROM DUAL;  SELECT TRUNC (12 . 1) FROM DUAL;  SELECT TRUNC (12 .56, 1) FROM DUAL;  SELECT TRUNC (12 .54, 1) FROM DUAL;  SELECT TRUNC (12 . 549, 2) FROM DUAL; |
| Example 5.12c (Ceil)  Returns the ceiling value (next highest integer above a number). The syntax for the ceil function is: ceil( number )  SELECT CEIL (12.5) FROM DUAL;  SELECT CEIL (12.1) FROM DUAL;  SELECT CEIL (13. 9) FROM DUAL; | 5.12d Floor  SELECT FLOOR (12.5) FROM DUAL,•  SELECT FLOOR (12 . 1) FROM DUAL,•  (13.9) FROM DUAL | 5.13Arithmetic  Simple arithmetic operations, such as multiplication (\*), division (/), addition (b), and subtraction (-), can be used in the SELECT clause of a query. Keep in mind that Oracle adheres to the standard order of operations: 1. Moving from left to right in the arithmetic equation, any required multiplication and division operations are solved first. 2. Addition and subtraction operations are solved after multiplication and division, again moving from left to right in the equation. To override this order Of operations, you can use parentheses to enclose the portion of the equation that should be calculated first. |
| Example 5.13a (Basic math)  —Does not change the underlying table,  SELECT salary, salary +200, salary/ 2, salary 2, salary-200 FROM patient; | Example 5.13b (Mod)  The MOD (modulus) function returns only the remainder of a division operation.  SELECT salary, MOD (salary, 2) FROM patient | **Example 5.13c (ABS)**  **The ABS (absolute) function returns the absolute, or positive, value of the numeric values supplied as the argument.**  **SELECT ABS (O) FROM DUAL;**  **SELECT ABS (10) FIROM DUAL;**  **SELECT ABS (-10) FROM DUAL;** |
| **Exmnple 5.13d (Pojver)**  **The POWER function raises the number in first argument to the power indicated as the second argument. The syntax of the POWER function is POWER( x, y), where x represents the number you're raising, and y represents the power to which you're raising it.**  **SELECT POWER (5, 3) FROM DUAL;**  **SELECT POWER (5, 0) FROM DUAL;** | **Example 5.13e (SORT)**  **The SQRT function returns the square root of a number**  **SELECT SQRT (10) FROM DUAL;** | **5.13 GREATEST, LEAST**  **The greatest function returns the greatest value in a list of expressions. The syntax for the greatest function is: greatest( exprl, expr2, ... expr\_n ) where exprl, expr2, .expr\_n are expressions that are evaluated by the greatest function. If the datatypes of the expressions are different, all expressions will be converted to whatever datatype exprl is.**  **The least function returns the smallest value in a list of expressions.The syntax for the least function is: least( exprl, expr2, ... expr\_n ) where exprl, expr2, .expr\_n are expressions that are evaluated by the least function. If the datatypes of the expressions are different, all expressions will be converted to whatever datatype exprl is.**  **SELECT GREATEST (10, 60, 90, 3) FROM DUAL;**  **SELECT LEAST (10, 60, 90, 3) FROM DUAL;** |
| Example 5.15a (Months\_betyceen)  The syntax is MONTHS\_ BETWEEN( dl, d2), where dl and d2 are the two dates in question, and d2 is subtracted from dl.To eliminate the decimal portion of the output and return only the number of whole months between the two dates, you can nest the MONTHS\_ BETWEEN function inside the TRUNC function,  SELECT DOB, MONTHS BETWEEN (SYSDATE, DOB) FROM patient;  SELECT DOB, TRUNC (MONTHS BETWEEN (SYSDATE, DOB) ) FROM patient;  --Can also find out the years by dividing the results of months\_between into 12.  SELECT DOB, TRUNC (MONTHS BETWEEN (SYSDATE, DOB) / 12) FROM patient;  SELECT DOB, ROUND (MONTHS BETWEEN (SYSDATE, DOB) / 12) FROM patient; | Example 5.15b (Add\_monlhs)  The syntax of the ADD\_ MONTHS function is ADD\_ MONTHS( d, m), where d rep-resents the beginning date for the calculation, and m represents the number of months to add to the date  SELECT DOB, ADD MONTHS (DOB, 4) FROM patient;  SELECT DOB, ADD MOUTHS (DOB, -14) FROM patient; | Example 5.15c (Next\_ day)  The syntax of the NEXT\_ DAY function is NEXT\_ DAY( d, DAY), where d represents the starting date, and DAY represents the day of the week to identify.The LAST\_ DAY function is similar to the NEXT\_ DAY function, except it always determines the last day of the month for a given date.  --The second argument represents the day as follows: --Sunday=1, Monday=2, Tuesday-3, ...  SELECT DOB, NEXT DAY (DOB, 4) TO CHAR (DOB, ' DI" ) TO CHAR (NEXT DAY (DOB, 4) , DY t ) FROM patient;  SELECT DOB, LAST DAY (DOB) DOB FROM atient; |
| Example 5.15d (Round)  SELECT sysdate FROM dual;  SELECT ROUND (sysdate, 'YEAR') FROM dual;  SELECT ROUND (sysdate, 'MONTH') FROM dual;  SELECT ROUND (TO DATE ( '27-oct-2011') , 'YEAR') FROM dual; SELECT ROUND (TO DATE ('27-JAN-2011') , 'YEAR' ) FROM dual;  SELECT ROUND ,TO\_DATE(’27-OCT-2011’) 'MONTH') FROM dual; | E.vmnple 5.15e (Greatest, Least)  5El,EC'I' gysdate, sysdate+5, sysdate-4 FROM dual;  SELECT GREATEST (sysdate, sysdate+5, sysdate-4) FROM dual;  SELECT LEAST (sysdate, sysdate•5, sysdate—4) FPC)M dual; | **5.15 NVL and NVL2 Functions**  **You can use the NVL function to address problems caused when performing arithmetic operations with fields that might contain NULL values. When a NULL value is used in a calculation, the result is always a NULL value.**  **The NVL function is used to substitute a value for the existing NULL so that the calculation can be completed. The syntax of the NVL function is NVL(x, y), where y represents the value to substitute if x is NULL In many cases, the substitute for a NULL value in a calculation is zero (O).** |
| Example 5.16a (NVL)  DELETE FPOM patient;  INSERT INTO patient values (211, 'john' ' Doe ' , 'm' , 'Il-FEB-1978 ' , NULL, Davis' 'CA') ;  INSERT INTO patient values (219, 'Billy' , 'smith' , 'f' 19-FEB-1998' , 2000, ' Sacramento' , ' CA') ;  SELECT fname, salary FROM patient;  --Every salary that is NULL IS replaced with zero.  SELECT fname, NVL (salary, O) FROM patient;  —Every salary that is NULL is replaced by the text (poor). Notice that to\_char is used --because both arguments have to be the same type (text).  SELECT fname, WL(TO CHAR (salary) , 'poor' ) "SALary" FROM patient; | Example 5.16a (NVL2)  The NVL2 function is a variation of the NVL function with different options based on whether a NULL value exists. The syntax of the NVL2 function is NVL2( x, y, z), where y represents what should be substituted if x isn't NULL and represents what should be substituted if x is NULL. This variation gives you a little more flexibility when working with NULL values.  —NVL2 provides a value if it not NULL and also provides a value if it is NULL. In this case, --poor if there is data and rich if it is NULL.  SELECT fname, salary, UVL2 (TO CHAR (salary) , 'poor' , rich' ) FROM patient; | 5.16 DECODE  The DECODE function takes a specified value and compares it to values in a list. If a match is found, the specified result is returned. If no match is found, a default result is returned. If no default result is defined, a NULL is returned. The syntax of the DECODE function is DECODE( V, L 1, RI, L2, R2, ..., D) where V is the value you're searching for, Ll represents the first value in the list, RI represents theresult to return if Ll and V are equivalent, and so on, and D is the default result to return if no match is found.  INSERT INTO patient values (311, 'Jakey' , 'Doer' , 'U' , ' Il-FEB-1978 ' , NULL, Davis'  --Notice there are an odd number of arguments.  --lf gender='m' then display Male  --else if gender='f' then display Female  SELECT gender, DECODE (gender, 'MALE ' f' , 'FEMALE' ) FROM patient;  --Notice there are an even number of arguments. The last argument pertains to the else condition.  --1f gender='m' then display Male  --else if gender='f' then display Female else display unknown  SELECT gender, DECODE (gender, 'm' , 'MALE' FEMALE UNKNOWN' ) FROM patient; |
| 5.17 SIGN  The syntax for the sign function is: sign (number )  number is the number to test for its sign.  If number < 0, then sign returns -1.  If number = O, then sign returns 0.  If number > 0, then sign returns 1.  DELETE FROM patient;  INSERT INTO patient values (111, 'john' , Doe ' , ' 11-FEB-1978 ' ,25000, Davis ' CA ' ) ;  INSERT INTO patient values (112, 'john' , ' Smith' , 'm' , '01-MAR-1981' , 40000, Davis ' CA' ) ;  INSERT INTO patient values (113, 'jill ' , 'Crane' , 'm' '12-APR1999' , 500000, 'Reno' , 'NV') ;  SELECT salary, DECODE (SIGN (salary-40000) , 0, 'Good money' , —1, 'Need more' , 'Donate' ) FROM patient;  SELECT salary, DECODE (SIGN (salary-40000) , 0, 'Good money ' , —1, 'Need more' , 1, 'Donate ) FROM patient; | 5.18 CASE  CASC expression syntax is similar to an IV-THEN-CLSC statement. Oracle checks each condition starting (rom the first condition (left to right). When a particular condition is satisfied (WHEN part) the expression returns the tagged value (THEN part). If none of the conditions are matched, the value mentioned in the CLSC part is returned. The ELSE part of the expression is not mandatory. CASE expression will return NULL if nothing is satisfied,  case when <condition> then evalue> then <value>  else<value> end  fname, salary,  CAZE WHEII THEN ' Need More'  WHEN lary-40000 THEN 'Okay'  Donate '  END  FROM atienL; | 5.19 TO NUMBER  The TO\_ NUMBER function converts a value to a numeric data type, if possible. For example, the string value 2009 stored in a date or character string could be converted to a numeric data type to use in calculations. If the string being converted contains non-numeric characters, the function returns an error. It just so happens that in the following cases, it does not matter if a conversion is made because ORACLE is smart enough to make the conversion by itself.  SELECT '2009' 2 FPC)M DUAL;  SELECT TO CHAR (2009) 2 FROM DUAL;  SELECT TO NUMBER ( '2009' ) 2 FROM DUAL; |
| 6.2 Dates  Sometimes you need to use a date as a search condition. Oracle displays dates in the default format DDMON-YY, with MON being the standard three- letter abbreviation for the month. Because the date field contains letters and hyphens, it's not considered a numeric value when Oracle performs searches.  Therefore, a date value must be enclosed in single quotation marks.  --When filtering for dates, can use the normal operators.  SELECT fname, Iname, DOB, salary FROM patient WHERE DOB>' Il-FEB-1978'  --1f the date format is not default, then to\_date has to be used.  SELECT fname, Iname, DOB, salary FROM patient WHERE DOB>TO DATE ( '02/11/1978' , t m/dd/yyyy') ; | 6.3 LOWER, UPPER  The upper function converts all letters in the specified string to uppercase. If there are characters in the string that are not letters, they are unaffected by this function. The syntax for the upper function is:upper( stringl )stringl is the string to convert to uppercase.  The lower function converts all letters in the specified string to lowercase. If there are characters in the string that are not letters, they are unaffected by this function. The syntax for the lower function is: lower(stringl) stringl is the string to convert to lowercase.  --These examples are to indicate that the data in the table is case sensitive.  SELECT fname, 1 name, city FROM patient WHERE city—' davist ; SELECT fname, 1 name, city FROM patient WHERE city—' Davis' ;  --To bypass case-sensitivity, use the upper or lower function. When using  —upper, make sure that it is being compared with something that is in upper case and the --opposite when checking for lower case.  SELECT fname, Iname,city FROM patient WHERE UPPER (city) -'DAVIS' •  SELECT fname, 1 name, city FROM patient WHERE LOWER (city) ='davis' • | Example 6.4a (The and clause)  At times, you need to search for records based on two or more conditions. In these situations, you can use logical operators to combine search conditions. The logical operators AND and OR are commonly used for this purpose. ( The NOT operator is also a logical operator in Oracle , but it's used to reverse the meaning of search conditions rather than combine them.) Keep in mind that when a query executes, records can be filtered with WHERE clause conditions. If the condition is TRUE when compared with a record, the record is included in the results. When the AND operator is used in the WHERE clause, both conditions combined by the AND operator must be evaluated as  TRUE, or the record isn't included in the results.  Can use the and operator to check for two conditions. Checks for a range in this case.  SELECT fname, Iname, DOB, salary FROM patient WHERE salary>30000 and gal ary <80000;  SELECT fname, Iname, DOB, salary FROM patient WHERE DOB>'11-FEB-1978' and DOB<' 15-MAR-1990' ;  SELECT fname, Iname, DOB, salary, city FROM patient WHERE salary>30000 and LOWER (city) = ' davis' ;  SELECT fname, Iname, DOB, salary, city FROM patient WHERE LOWER (city) —'davis' and LOWER (city) z' reno' ;  —What is wrong?? Notice that the ranges overlap which makes it logically invalid.  SELECT fname, Inane, DOB, salary FROM patient WHERE salary>30000 and salary >80000; |
| Example 6.4b (Betrveen)  The two values defining the range for SQL BETWEEN clause can be dates, numbers or just text. (Inclusive of the end points)  --When looking for a range using the and clause, if it is inclusive of the end points then it -- is easier to use the between clause instead.  SELECT fname, Inane, DOB, salary FROM patient WHERE AND  salary  SELECT fname, Iname, DOB, salary FROM patient WHERE salary BETWEEN 30000 AND 80000; | Example 6.5a (Or, In)  The IN o erator allows ou to s ecif multi le values in a WHERE clause.  --lf the same column is being checked against multiple values, then the IN clause is an easier route.  SELECT fname, Inane, city FROM patient WHERE city—' Davis' OR city=' Sacramento' OR city= ' Chico' ;  —Same as above  SELECT fname, Iname, city FROM patient WHERE city IN  ( 'Davis' , 'Sacramento' , 'Chico') ;  SELECT fname, Jname, salary FROM patient WHERE salary=10000 OR salary=20000 OR salary=30000;  SELECT fname, 1 name, salary FROM patient WHERE salary (10000, 20000, 30000) ;  --Negates the IN clause using the NOT operator which means it displays the record as  —long as the salary is not one of the three items on the list.  SELECT fname, Iname, salary FROM patient WHERE salary NOT IN (10000, 20000, 30000) ; | Example 6.5b (Order ofprecendence)  Next, take a look at the order of logical operators. Because the WHERE clause can contain multiple types of operators, you need to understand the order in which they're resolved: t?] Arithmetic operations are solved first. Comparison operators are solved next. Logical operators have a lower precedence and are evaluated last in the order NOT, AND, and OR.  -- The AND clause is gotng to be done first and then the OR clause. This means it first  --100ks at the salary range and comes back with a result. If the salary is in the range or  -the City Davis then the record is selected.  SELECT Inarne, DOB, salary, city FROM patient WHERE scalar OR city='Davio t ; AND |
| Example 6.6a (Any)  -- Since the values overlap, the following is logically not correct.  SELECT fname, 1 name, salary FROM patient WHERE salary > 30000 OR salary>40000;  --This accomplishes the same thing as above.  SELECT fname, Inane, salary FROM patient WHERE salary 40000) ; | **Example 6.6b (All)**  **Compares a value to every value in a list or returned by a query. Must be preceded by =, > , < , Evaluates to TRUE if the query returns no rows**  **--The salary has to be greater than both of those values. Once again is logically not correct.**  **SELECT fname, Inane, salary FROM patient WHERE salary > 30000 AND salary>40000;**  **--Same as above**  **SELECT fname, Iname, salar FROM atient WHERE salar >ALL (30000, 40000) ;** | **6.8 NULL**  **When performing arithmetic operations or search conditions, NULL values can cause unexpected results. A NULL value means no value has been stored in that field. Don't confuse a NULL value with a blank space. A NULL is the absence of data in a field; a field containing a blank space does contain a value— a blank space. When searching for NULL values, you can't use the equal sign because there's no value to use for comparison in the search condition. When checking for a NULL value, you're actually checking the status of the column: Does data exist or not? If you need to identify records that have a NULL value, you must use the IS NULL comparison operator.**  **INSERT INTO patient values (978, 'john' , 'Doe' , ' m', '11-FEB1978' , 25000, NULL, 'CA') ;**  **SELECT fname, 1 name,city FROM patient;**  **--Not correct. When checking for NULL , the (IS) keyword must be used.**  **SELECT fname, Iname,city FROM patient WHERE city= NULL;** |
| 6.9 Creating tables with select statements  Example 6.9a (Creating table using select)  --Can feed the results to another table instead of displaying it to the screen. The new —table (patient\_temp) will be comprised of fname, Iname, salary. This new table will --contain the data from the patient table.  --Keep in mind that the column names must be valid identifiers, which means that things —such as calculations or function names which have parentheses in them will not be  --accepted. For this reason, an alias is used. SELECT fname, cit from atient;  CREATE TABLE patient temp AS SELECT fname, Iname, salary, city FROM patient WHERE city— 'Mavis';  SELECT \* FROM patient temp; | E.vmnple 6.9b (Creating tables and aliases)  --Creating a table and using an alias name  'TABLE patient \_ temp;  —INVALID: have to use an alias.  CREATE TABLE patient temp SELECT fname, Inane, salary\*2 FROM patient WHERE salary >30000;-  CREATE TABLE patient temp AS SELECT fname, Inarne, sa lary\*2 new salary FROM patient WHERE salary >30000;  —Notice the column name new\_salary.  DESC patient tem | **Example 6.9c (Creating table with constraints)**  **In creating a copy, only the not NULL constraint gets copied over. Also the name of the constraint will be different for the new table**  **DPOP TABLE test;**  **DROP TABLE test2;**  **COLUMN search condition FOPMAT AIO**  **CREATE TABLE**  **coi I NUMBER PP. IMAPY K FTI, co] 2 UUMBEP (JUIOUF„, col 3 UUMBEP CHECK (co] 3>20) ,**  **NUMBER NOT NULL** |
| **6.10 Updating tables using the update statement**  **The UPDATE statement allows you to update a single record or multiple records in a table.**  **The syntax for the UPDATE statement is:**  **UPDATE table SET column = expression WHERE predicates;**  **143**  **Example 6.10a (Updating)**  **SELECT fname, 1 name, salary FROM patient where patient id—l Il;**  **—Updates the patient table by setting the fname, Iname and salary for patient (11). UPDATE patient SET fname-'Bi11' Iname=' Bob' , salary—10000 WHERE patient id=lll;**  **--Confirm results.**  **SELECT fname, I name, salary FROM patient where patient id=lll;** | **Example 7a (Order by)**  **The ORDER BY clause, used to display query results in a sorted order, is listed at the end of the SELECT statement,The columns used to sort the results are listed in the ORDER BY clause.ln the query results, the second column ( Name) is listed in ascending alphabetical order. Note these important points:**  **--Default sort is in ascending order.**  **SELECT patient id, fname, Iname, city CITYNAME FROM patient ORDER BY city;**  **--ASC is implied by default.**  **SELECT patient id, fname, Iname, city CITYNAME FROM patient ORDER BY city ASC; --Can sort in descending order.**  **SELECT patient id, fname, Iname, city CITYNAME FROM patient ORDER BY city DESC;**  **--Can sort by the position 01 the column between the select and from clause.**  **SELECT patient \_ 1.1, Jname, city CJTYNAME, FROM patient ORDER BY 4;**  **—Can use the alias name for sorting.**  **SELECT CITYNAME FROM patient ORDER BY CITYNAME;** | Example 7b (NULLs)  When sorting in ascending order, values are listed in this order:  1. Blank and special characters  2. Numeric values  3. Character values ( uppercase first)  4. NULL values  Unless you specify " DESC" for descending, the ORDER BY clause sorts in ascending order by default.lf a column alias is given to a field in the SELECT clause, you can reference the field in the ORDER BY clause with the column alias— although doing so isn't required.You can also use the ORDER BY clause with the optional NULLS FIRST or NULLS LAST keywords to change the order for listing NULL values. By default, NULL values are listed last when results are sorted in ascending order and first when they're sorted in descending order.  --NULLs appear last by default. You can change the default to make them appear first.  SELECT patient id, fname, Iname, city CT TYNAME FROM patient ORDER BY city  NULLS EIR5T;  --NULLs appear last which is implied.  SELECT patient id, fname, Iname, city CITYNÄME FROM patient ORDER BY city NULLS LAST; |
| **Example 7c (Secondary sorts)**  **In the previous examples, only one column was specified in the ORDER BY clause, which is called a primary sort. In some cases, you might want to include a secondary sort, which specifies a second field to sort by if an exact match occurs between two or more rows in the primary sort.**  **149**  **--0rder% by fname, When it comes across, fnames that are the same, then it further sorts •by Iname. Both sorts are in ascending order.**  **SELECT patient id, fname, Iname,city CITYNAME FROM patient ORDER BY fname, 1 name ;**  **--Can make fname in descending and Iname in ascending order which is default.**  **SELECT patient\_id, fname, 1 name, city CITYNAME FROM patient ORDER BY fname DESC 1 name ;** | **Example 7d (Position)**  **Oracle also provides an abbreviated method for referencing the sort column if the name is used in the SELECT clause. In the previous example, State and City are used in both the SELECT and ORDER BY clauses. Instead of listing these column names again in the ORDER BY clause, you can reference them by their positions in the SELECT clause's column list. You can also use the column alias.**  **--Can use the order by on the actual column, the position of where it appears between —the select and from or by the alias.**  **SELECT patient id, fname, Inarne, city CITYNAME FROM patient ORDER BY 4;**  **SELECT patient\_id, fname, 1 name, city CITYNAME FROM patient ORDER BY CITYNAME;** | **Example 7e (column versus alias)**  **Difference in ordering by the column versus by the alias**  **--1n this dataset there is one record that has a NULL for salary.**  **--0rder by alias after zero has been substituted.**  **SELECT WI. (salary, O) pay FROM patient ORDER BY pay;**  **--0rder by position after zero has been substituted.**  **SELECT (salary, O) FROM patient ORDER BY 1;** |
| **8.1 Grou b exam les**  **SELECT SUM (salary) FROM patient;**  **SELECT cit SUM (salar ) FROM atient GROUP BY city;**  **SELECT state, SUM(sa1ary) FROM patient GROUP BY state;**  **SELECT gender, SUM (salar ) FROM patient GROUP BY gender;**  **ELECT city, gender, SUM (salary) FROM patient CROUP BY cit , gender;**  **SELECT state, gender, SUM (salary) FROM patient GROUP BY state, gender;**  **SELECT fname, city, SUM(sa1ary) FROM patient GROUP BY fname, city;** | **8.2 SUM**  **The SUM function is used to calculate the total amount stored in a numeric field for a group of records. The syntax of the SUM function is SUM(([ DISTINCTI ALL] n), where n is a column containing numeric data.**  **SELECT salary from patient;**  **—The result is a single number that is the summation of all the salaries. SELECT SUM (salary) FROM patient;**  **--AII is implied as in the above statement.**  **SELECT SUM (ALL salary) FROM patient;**  **--The result is a single number that is the summation of all the distinct salaries, which --means it suppresses the duplicates before doing a summation.**  **SELECT SUM (Dls rrmcrr salary) FROM patient;**  **Creates a grouping for each city, which means that it suppresses the duplicates in each group and --and then comes up with a summation for each group order by is the last clause.**  **SELECT city, SUM (salary) FROM patient GROUP BY city ORDER BY 1;**  **--Creates a group for each of the states and gives a summation for each state.**  **SELECT state, SUM (salary) FROM patient GROUP BY state;**  **--Creates combination of fname, city catagories and provides a summation for each group.**  **SELECT fname, cit SUM (salar ) FROM patient GROUP BY fname**  **--First the where clause filters. Then it does a grouping with the data that is left over. It**  **--groups the different cities and then for each city group, it comes up with a summation.**  **SELECT city, SUM (salary) FROM patient WHERE UPPER 'RENO' BY city ORDER BY 1; GROUP**  **--First the where clause filters. Then it does a grouping with the data that is left over. It**  **—groups the different cities and then for each city group, it comes up with a summation.**  **SELECT city, SUM (salary) FROM patient WHERE UPPER (city) 'RENO' or city is NULL GROUP BY cit ORDER BY 1;** | **8.3 DISTINCT**  **The optional DISTINCT keyword instructs Oracle to include only unique numeric values in its calculation. The ALL keyword instructs Oracle to include multiple occurrences of numeric values when totaling a field. If the DISTINCT or ALL keywords aren't included when using the SUM function, Oracle assumes the**  **ALL keyword by default and uses all the numeric values in the field when the query is executed.**  **--ALL is implied and is not needed.**  **SELECT SUM (ALL salary) FROM patient;**  **--Suppresse5 the duplicates and gives a summation.**  **SELECT SUM (DISTINCT salary) FROM patient;**  **--Suppresses duplicates and gives a summation for each city category.**  **SELECT city, SUM (DISTINCT salary) FROM patient GROUP BY city**  **--Does not suppress the duplicates for salary and gives a summation for each city category.**  **SELECT city, SUM (ALL salary) FROM atient GROUP BY cit** |
| **8.4 AVG**  **The AVG function calculates the average of numeric values in a specified column. The syn-tax of the AVG function is AVG([ DISTINCTI ALL] n), where n is a column containing numeric data.**  **SELECT salary FROM patient;**  **--Gives a single average for all the salaries.**  **SELECT AVG (salary) FROM patient;**  **—Same as above**  **SELECT AVG (ALL salary) FROM patient;**  **—Suppresses duplicates and then gives an average.**  **SELECT AVG (DISTINCT salary) FROM patient;**  **--lnvalid: Does not know how to display the one single average salary with all the cities. SELECT city, AVG (salary) FROM patient;**  **--Displays the average salary for each city category.**  **SELECT city, AVG (salary) FROM patient GROUP BY city ORDER BY 1;**  **--First it filters the data based on the where clause. Then it takes the left over records and --does a grouping for each of the cities and provides an average for each grouping.**  **SELECT city, AVG (salary) FROM patient WHERE UPPER (city) <> 'RENO' GROUP BY city ORDER BY 1;**  **—First it filters the data based on the where clause. Then it takes the left over records and**  **--does a grouping for each of the cities and provides an average for each grouping. If there is a**  **--city group that does not have a salary, which means that it is NULL, then it will replace it with a**  **--zero. There is additional filtering using the having clause after all the grouping is done,**  **SELECT city, AVG (nvl (salary, 0) ) FROM patient WHERE UPPER (city) <> 'RENO'**  **GROUP BY city HAVING AVG (salary) >20000 ORDER BY 1;** | **8.5 COUNT**  **Depending on the argument used, the COUNT function can count the records having non- NULL values in a specified field or count the total records meeting a specific condition, including those containing NULL values. The syntax of the COUNT function is COUNT(\* [ DISTINCTI ALL] c), where c represents a numeric or non-numeric column.**  **SELECT fname, 1 name, city FROM patient;**  **--Counts the number of rows.**  **SELECT COUNT (+) FROM patient;**  **--Counts the number rows based on the contents of the city. If the city for a given row contains --a NULL, then it will not be counted.**  **SELECT CO(JWr (city) FROM patient;**  **--Same as above**  **SELECT COUNT (ALL city) FROM patient;**  **--lnvalid: Does not know how to display a single number with the six different cities.**  **SELECT city, COUNT ( \* ) FROM patient;**  **--Create a group for each of the different cities and do a count for each category. NULL cities are —excluded from the count.**  **SELECT city, COUNT (city) FROM patient GROUP BY city;**  **—Same as above but the NULLs are not excluded.**  **SELECT city, COUNT ( w ) FROM patient GROUP BY city;**  **--After it has come up with the count per grouping, there is an additional filtering, which includes only those --records where the count is greater than I.**  **SELECT city, COUNT FROM patient GROUP. BY city HAVING COUNT** | **8.6 MAX**  **The MAX function returns the largest value stored in the specified column. The syntax of the MAX function is MAX([ DISTINCTI ALL] c), where c can represent any numeric, character, or date column.**  **SELECT salary FROM patient;**  **--The highest salary is displayed.**  **SELECT MAY. (salary) FROM patient; --same as above**  **SELECT MAX (ALL salary) FROM patient;**  **--lnvalid• the highest salary is a single number and cannot be associated with all cities. SELECT city, MAX (salary) FROM patient;**  **--Displavs highest salary for each city.**  **SELECT city, MAX (salary) FROM patient GROUP BY city;**  **—Given the fname, city combination, display the the number of records and the highest salary —for each of those combination categories.**  **SELECT fname, city, AVG (salary), MAX (salary) FROM patient GROUP BY fname, city;**  **—Same as above except that after the final result, do some additional filtering based on the count.**  **SELECT fname, city, AVG (salary) , MAX (salary) FROM patient GROUP BY fname, city HAVING COUNT (+) > 2;** |
| **8.7 MIN**  **In contrast to the MAX function, the MIN function returns the smallest value in a specified column. As with the MAX function, the MIN function works with any numeric, character, or date column. The syntax of the MIN function is MIN([ DISTINCTI ALLJ c), where c represents any character, numeric, or date column. The MIN function uses the same logic as the MAX function for numeric and character data, except it returns the smallest value rather than the largest value.**  **SELECT salary FROM patient;**  **--The lowest salary is displayed.**  **SELECT MIN (salary) FROM patient;**  **—Invalid: cannot display a single number with six cities.**  **SELECT city, MIN (salary) FROM patient;**  **—Display the lowest salary for each city category and display the number of records in each group.**  **SELECT city, MIN (salary) , COUNT (O FROM patient GROUP BY cit**  **-Jilters with the where clause. Then given the remaining records, it groups by city and finds the**  **•lowest salary lor each city category. Given the result set, it only includes the ones where -there are more than two records for each group. The results are sorted by city.**  **SELECT city, MIN (salary) FROM patient WHERE city IS NOT NULL GROUP BY city HAVING ORDER BY 1 DESC;** | **8.8 Dates and group functions**  **—Displays oldest person, youngest person, the number of records (excludes all those that have a NULL in --DOB), and number of records (Suppresses duplicate DOB).**  **SELECT min (DOB) , max (DOB) , count (DOB) , count (DISTINCT DOB) FROM patient;**  **--lnvalid: Cannot apply AVG to date formats. I-Ise months\_between to convert it into a number —and then do an average.**  **SELECT AVG (DOB) FROM patient;**  **--lnvalid: can do a sum on date formats.**  **SELECT SUM (DOB) FROM patient;** | **Example 9c (Using subqueries-one single row)**  **A single- row subquery can return only one row of results consisting of only one column to the outer query. A single- row subquery can also be nested in the outer query's SELECT clause.**  **—Display all the patients who have malaria. Returns a single row.**  **—Start with inner most query and work your way to the outer query. Notice the number of —parantheses. The indentation is used for readability.**  **SELECT fname, 1 name FROM patient WHERE (**  **SELECT patient id FROM patient\_disease WHERE disease SELECT¯disease id FROM disease WHERE disease desc='Ma1aria') ) ;**  **--lnvalid. Notice the asterisk does not match up with disease\_id.**  **—The disease id must match with disease\_id and not the asterisk.**  **SELECT fname, Iname FROM patient WHERE patient (**  **SELECT patient id FROM patient disease WHERE disease (**  **SELECT FROM disease¯WHERE disease desc='Ma1aria') ) ;**  **--lnvalid. Notice the asterisk does not match up with patient\_id.**  **SELECT fname, Inarne FROM patient WHERE patient**  **SELECT FROM patient disease WHERE**  **SELECT disease\_id FROM disease WHERE disease desc—'Ma1aria' ) ) ;** |
| **Example 9d (Multiple rows)**  **Multiple- row subqueries are nested queries that can return more than one row of results to the parent query. The main rule to keep in mind when working with multiple- row subqueries is that you must use multiple- row operators. If a single- row operator is used with a subquery that returns more than one row of results, Oracle returns an error message. Valid multiple- row operators include IN, ALL, and ANY must be used. Of the three, the IN Operator is used most often.**  **--lnvalid. Display all the patients who have Cancer. Returns multiple rows.**  **--Start from the inner-most query. The result would then bubble up to the outer queries. The**  **--problem with this query is that there are multiple patients who suffer from cancer. This**  **—would mean that the second subquery would return multiple rows; however, the from the outer —most query can only handle one single piece of information.**  **SELECT fname, Iname FROM patient WHERE (**  **SELECT patient\_id FROM patient disease WHERE disease**  **SELECT disease\_id PROü disease WIIERE 'Cancer' ) ) ;**  **—To correct the above problem, we change from (z) to (in). The in operator can handle multiple values.**  **SELECT fname, Iname FPC)M pat j ent WHERE patient id**  **SELECT patient id FP.OM patient \_disease WHERE disease ide (**  **SELECT disease id FROM disease WHERE disease desc—'Cancer' ) ) ;** | **Example 9e (Single and multiple rows)**  **—Display all the diseases that "jill crane" has.**  **SELECT disease desc FROM disease WHERE disease**  **SELECT disease\_id FROM patient disease WHERE (**  **SELECT patient id FROM patient WHERE fname—' j ill' and Iname=' Crane' ) ) ;**  **--lnvalid. Display all the diseases that "John Doe" has. Must use in clause.**  **SELECT disease desc FROM disease WHERE disease id —**  **SELECT disease id FROM patient disease WHERE patient (**  **SELECT patient id FROM patient WHERE fname=' john' and Iname=' Doe' ) ) ;**  **SELECT disease desc FROM disease WHERE disease id IN (**  **SELECT disease id FROM patient disease WHERE patient id= (**  **SELECT patient id FROM patient WHERE fname—' john' and Inane—' Doe' ) ) ;**    **--Notice the concatenation operator in the subquery.**  **--The concatenation operator takes the two pieces of data and connect them together --to make it appear as one single piece.**  **SELECT disease desc FROM disease WHERE disease id IN (**  **SELECT disease id FROM patient disease WHERE, patient id= (**  **SELECT patient\_id FROM patient WHERE (fname I I 'johnDoe') ) ) ;** | **Example 91 (Multiple column subquery)**  **Multiple- column subquery returns more than one column to the outer query. The syntax of the outer WHERE clause is WHERE ( columnname, columnname, ...) IN subquery.**  **Keep these rules in mind: Because the WHERE clause contains more than one column name, the column list must be enclosed in parentheses. Column names listed in the WHERE clause must be in the same order as they're listed in the subquerVs SELECT clause.**  **DROP TABLE special names;**  **\_**  **CREATE TABLE special names (fname VAPCHAR2 (20) , Inarne VARCHAR2 (30) ) ; INSERT INTO special names VALUES ( 'john ' , ' Doe' ) ;**  **\_**  **INSERT INTO special names VALUES ( 'j ill' , 'Crane' ) ;**  **--Notice that fname and Iname are enclosed in parantheses in the outer query. It works**  **--like the concatenation operator in that fname and Iname become a single piece of data —which are compared against fname and Iname in the inner query. The (IN) operator is used --because it is possible that the inner query may yield multiple rows.**  **SELECT patient id FROM patient WHERE (fname, Iname) IN (**    **SELECT fname, I name FROM special names) ;** |
| **Example 9g (Groupfunctions and subqueries)**  **SELECT AVG (salary) FROM patient;**  **--lnvalid. Must use a subquery. For every row that is processed from the patient table —we have to compare its salary against the AVG(salary). We cannot combine a group —function with row level processing which is why this gives us an error.**  **SELECT fname, 1 name, salary FROM patient WHERE salary > AVG (salary) ;\**  **—In this case, the inner query will be executed which comes up with a single number.**  **—That single number will be fed to the outer query which can be used to compare every row in the patient table.**  **SELECT fname, Iname, salary FROM patient WHERE salary > (SELECT AVG (salary) FROM patient) ;**  **--lnvalid: AVG cannot be used on DATE datatypes**  **SELECT fname, Iname, DOB FROM patient where DOB>**  **(SELECT AVG (DOB) FROM patient) ;**  **--To make AVG work, dates have to be converted to numbers which can be done by using**  **--MONTHS BETWEEN. Notice that a subquery has to be used to deal with the AVG first.**  **SELECT fname, Iname, DOB FROM patient where MONTHS BETWEEN (sysdate, DOB) > (SELECT AVG (MONTHS BETWEEN(s sdate, DOB) ) FROM patient) ;** | **Example 9h (Create table and subqueries)**  **You can also perform CREATE TABLE AS by using subqueries.**  **--lnstead of displaying the information on to the screen, it can be fed into a brand new table.**  **CREATE TABLE NEW TABLEI ms SELECT patient id FROM patient WHERE (fname, 1 name) IN (SELECT fname, 1 name FROM special \_names) ;**  **SELECT \* FPOM NEW TABLEI;**  **--lnvalid. Mur.t use an alias because the new table will be using the information between --the select and from to come up with the column names for the new tables. Since —salary' 2 is not a valid column name, an alias has to be used.**  **CREATE TABLE NEW TABLE2 AS SELECT patient id, salary \* 2 FROM patient WHERE (fname, Iname) IN (SELECT fname, I name FROM special \_names) ;**  **--This query corrects the problem from the previous example.**  **CREATE TABLE NEW TABLE2 AS SELECT patient id, salary**  **2 Increase FROM patient WHERE (fname, I name) IN ( SELECT fname, Inarne FROM special names) ;**  **SELECT \* FROM NEW TABLE2;** | **Exmnple "i (Update and delete using subqueries)**  **You can also perform UPDATE and DELETE statements**  **SELECT pat-ienL\_id, salary FROM patient;**  **--This example updates the salary for all those patients who have cancer.**  **UPDATE patient SET salary—salary "2 WIIERE patient id IN (**  **SELECT patient id FROM patient \_disease WHERE disease id— (**  **SELECT disease id FROM disease WHERE )**  **SELECT patient id, salary FROM patient;**  **—This example deletes all the records from the patient\_disease table for all those who have cancer.**  **DELETE FROM patient disease WHERE disease\_id IN (**  **\_**  **SELECT disease id FROM disease WHERE disease desc='Cancer') ;**  **SELECT \* FROM patient disease;** |

|  |
| --- |
| 0) Copy and paste the data from PoliticalCandidate.txt |
| 1) Display the records in the candidate table such that first name, last name and dob appear as one single column with a column alias named (full information). Place a comma as a separator between each field. Also make sure that you upper case the first letter of both the first and last name. (Use the concat function or the || symbols)  e.g. John,James,01 Feb 90 |
|  |
| 2) Display the name and age of each of the candidates. (You have to calculate the age using the various date functions based on date of birth) |
|  |
| 3) Display the last name of all the candidates. Make sure that no duplicate last names are displayed |
|  |
| 4) Display the first name of all the candidates who are greater than 40 years old |
|  |
| 5) Display the first name of all the candidates whose salary is null, as well as salaries that are between 20,000 and 40,000 |
|  |
| 6) Display the first name of all the candidates that begin with the character ‘C’ regardless of case and are older than 40 years old |
|  |

|  |
| --- |
| 1) Display the lastname, firstname and salary of everyone in the candidate table. Order in ascending order based on lastname.  Use the column name to sort |
|  |
| 2) Display the lastname, firstname and salary of everyone in the candidate table. Order in Descending order based on the combination of lastname and first name  Do not use the column names but rather their position |
|  |
| 3) Display the lastname, firstname and salary \* 2 (use alias double\_salary) of everyone in the candidate table. Order in Descending order based on the alias |
|  |

|  |
| --- |
| 1) Display the first name of all the candidates who are Democrats. (use subquery) |
|  |
| 2) Display the party description of the all the candidates whose last name ends with ‘C’ regardless of case. (Use subquery) |
|  |
| 3) Display the first name of all the candidates who are either Democrats or Republicans regardless of case. (use subquery, use in clause) |
|  |
| 4) Create a second table called candidate2 that contains all the data from the candidate table using create table as… statement. The new table should contain only the first name, lastname and the salary, lowered by 10% for all the candidates who are Republicans |
|  |
| Note: Display the party code, not the party description in each of the following |
| 1) Display the number of people in each party. Order by Party. (Make sure to display the party\_code) |
|  |
| 2) Display the number of people in each party whose first name starts with d or r. (Make sure to display the party\_code) |
|  |
| 3) Display the average salary for each party (Make sure to display the party\_code) |
|  |
| 4) Display the number of people in each party where the number of people does not exceed 2 |
|  |
| 5) Display the average salary for each party where the average does not exceed 50000 |
|  |
| 6) Create a new table called candidate2 that contains the number of people in each party. Should contain the partycode and the number of people (CAUTION, you have to use an alias for this to work) |
|  |

|  |  |  |
| --- | --- | --- |
| 0 |  | |
| 1 | | Using a single SQL statement display fname, lname, dob, salary for all the **students** whose age is greater than 15. (Have to convert the dob to years) |
|  | |  |
| 2 | | Using a single SQL statement display the following from the **student** table.  *ssn, lname and fname* concatenated together with a comma and a space separating the two **(e.g sabzevary, IRAJ).** The last name should be all lower case. The first name should be all upper case. The heading on the column should be **Full\_Name (Use the concat function or the || symbols)** |
|  | |  |
| 3 | | Using a single SQL statement display fname, lname, dob, salary from the **student** table where the lname contains the letters ‘h’ or ‘a’ regardless of case (Use the like clause) |
|  | |  |
| 4 | | Using a single SQL statement display fname, lname, dob, salary from the **student** table where age is between 15 and 25. (use the between clause) and fname starts with ‘abr’ regardless of case. If the dob is null, display ‘not born yet’ (USE NVL) |
|  | |  |
| 5 | | Using a single SQL statement display fname, lname, dob, salary from the **student** table where the dob is not null. If the salary is <20000 display ‘poor’ otherwise display ‘rich’ (Use decode) |
|  | |  |
| 6 | | Using a single SQL statement display the square root of dob plus 20 divided by 5 from the **student** table( CAUTION: The order of precedence is as the question is read. Use paranthesis) (Have to convert dob to years first) |
|  | |  |
| 7 | | Using a single SQL statement display fname, lname, dob, salary from the **student** table where the first name of the student can be **anything except** John, Jack or Bob. (Use the IN or NOT IN syntax) |
|  | |  |
| 8 | | Using a single SQL statement display fname, lname, dob, salary from the **student** table where the fname is only three characters long; the first character and second characters can be anything, but the third character must be ‘b’ (e.g. bob, cib, lib, hub, mob). Also the salary must be greater than 10000 and the phone number must start with ‘527’ |
|  | |  |
| 9 | | Create a new table called student2 that contains the results from the following SQL statement: fname, lname, salary\*2 from the **student** table where last name contains the letters ‘nn’ (e.g. Benny, Bonny, Sonny) and dob does not contain any data. (NOTE: Beware of salary\*2 for the create table statement) |
|  | |  |

|  |  |  |
| --- | --- | --- |
| 0 |  | |
| 1 | | Write a single SQL statement that displays the number of of people with the same lastname. The results should contain the lastname and the count for each lastname. Exclude from the list all those who live in CA |
|  | |  |
| 2 | | Write a single SQL statement that displays the number of people living in each of the states. The results should display the state and the number of people living in each state. Exclude from the list all those who are living in cities that contains the letter ‘h’ |
|  | |  |
| 3 | | Use a single SQL statement that displays the ssn and the number of classes a student is taking with the column heading “number of classes” where the number of classes is less than 2 , order by ssn descending. |
|  | |  |
| 4 | | Write a single SQL statement that displays the average age for each city, state combination for all students whose salary is greater than the average salary and are taking some kind of ‘Intro’ class. Also exclude the city ‘Berkeley’ from the list regardless of case. Sort by city in ascending order and state in descending order |
|  | |  |
| 5 | | Write a single SQL statement that displays the States in lower case along with the rounded average age for the different states with the alias name “average of ages” for all the students who are taking a class that contains ‘principles’ in its description regardless of case. |
|  | |  |