Set Operators

- Set operators are used to combine the results from different SELECT statements into one single result output.
- Sometimes you want a single output from more than one table.
- If you join the tables, the rows that meet the join criteria are returned, but what if a join will return a result set that doesn't meet your needs?
- This is where SET operators come in.
- They can return the rows found in multiple SELECT statements, the rows that are in one table and not the other, or the rows common to both statements.

- In order to explain the SET operators, the following two lists will be referred to throughout this lesson:

$$A = \{1, 2, 3, 4, 5\}$$

$$B = \{4, 5, 6, 7, 8\}$$

- Or in reality: two tables, one called A and one called B.

A

A_ID
1
2
3
4
5

B

B_ID
4
5
6
7
8

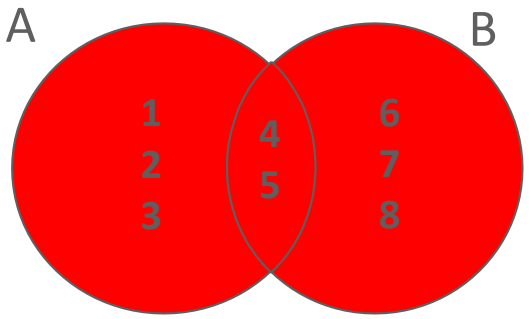
Rules to Remember

- There are a few rules to remember when using SET operators:
  - The number of columns and the data types of the columns must be identical in all of the SELECT statements used in the query.
  - The names of the columns need not be identical.
  - Column names in the output are taken from the column names in the first SELECT statement.
- So any column aliases should be entered in the first statement as you would want to see them in the finished report.

UNION

- The UNION operator returns all rows from both tables, after eliminating duplicates.

```
SELECT a_id
FROM    a
UNION
SELECT b_id
FROM    b;
```

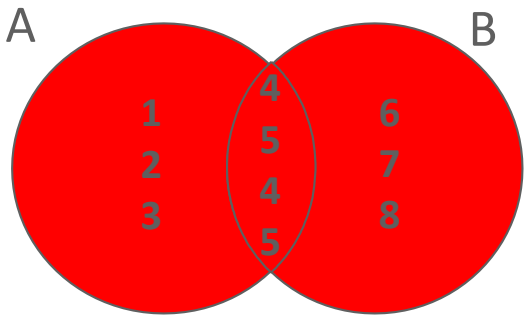


- The result of listing all elements in A and B eliminating duplicates is {1, 2, 3, 4, 5, 6, 7, 8}.
- If you joined A and B you would get only {4, 5}. You would have to perform a full outer join to get the same list as above.

UNION ALL

- The UNION ALL operator returns all rows from both tables, without eliminating duplicates.

```
SELECT a_id
FROM    a
UNION  ALL
SELECT b_id
FROM    b;
```

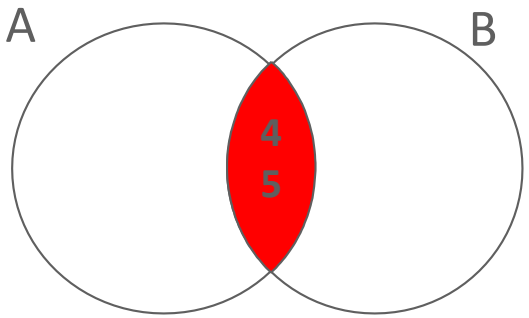


- The result of listing all elements in A and B without eliminating duplicates is {1, 2, 3, 4, 5, 4, 5, 6, 7, 8}.

INTERSECT

- The INTERSECT operator returns all rows common to both tables.

```
SELECT a_id
FROM    a
INTERSECT
SELECT b_id
FROM    b;
```

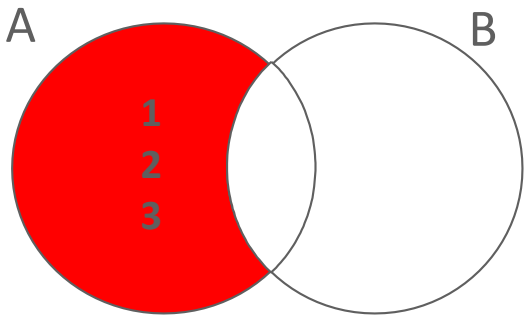


- The result of listing all elements found in both A and B is {4, 5}.

MINUS

- The MINUS operator returns all rows found in one table but not the other.

```
SELECT a_id
FROM    a
MINUS
SELECT b_id
FROM    b;
```



- The result of listing all elements found in A but not B is {1, 2, 3}.
- The result of B MINUS A would give {6, 7, 8}.

Set Operator Examples

- Sometimes if you are selecting rows from tables that do not have columns in common, you may have to create your own columns in order to match the number of columns in the queries.
- The easiest way to do this is to include one or more NULL values in the select list.
- Remember to give each one a suitable alias and matching data type.

Set Operator Examples

- For example:
  - The employees table contains a hire date, employee id and a job id.
  - The job history table contains employee id and job id, but does not have a hire date column.
  - The two tables have the employee id and job id in common, but job history does not have a start date .
- You can use the TO_CHAR(NULL) function to create matching columns as in the next slide.

Set Operator Examples

```
SELECT hire_date, employee_id, job_id
FROM employees
UNION
SELECT TO_DATE(NULL),employee_id, job_id
FROM job_history;
```

HIRE_DATE	EMPLOYEE_ID	JOB_ID
17/Jun/1987	100	AD_PRES
17/Sep/1987	200	AD_ASST
21/Sep/1989	101	AD_VP
03/Jan/1990	103	IT_PROG
21/May/1991	104	IT_PROG
13/Jan/1993	102	AD_VP
07/Jun/1994	205	AC_MGR
07/Jun/1994	206	AC_ACCOUNT
17/Oct/1995	141	ST_CLERK
17/Feb/1996	201	MK_MAN
11/May/1996	174	SA_REP
29/Jan/1997	142	ST_CLERK
17/Aug/1997	202	MK_REP
15/Mar/1998	143	ST_CLERK
24/Mar/1998	176	SA_REP
09/Jul/1998	144	ST_CLERK
07/Feb/1999	107	IT_PROG
24/May/1999	178	SA_REP
16/Nov/1999	124	ST_MAN
29/Jan/2000	149	SA_MAN
-	101	AC_ACCOUNT
-	101	AC_MGR
-	102	IT_PROG
-	114	ST_CLERK
-	122	ST_CLERK
-	176	SA_MAN
-	176	SA_REP
-	200	AC_ACCOUNT
-	200	AD_ASST
-	201	MK_REP

Set Operator Examples

- The keyword NULL can be used to match columns in a SELECT list.
- One NULL is included for each missing column.
- Furthermore, NULL is formatted to match the data type of the column it is standing in for, so TO_CHAR, TO_DATE, or TO_NUMBER functions are used to achieve identical SELECT lists.

SET Operations
ORDER BY

- If you want to control the order of the returned rows when using SET operators in your query, the ORDER BY statement must only be used once, in the last SELECT statement in the query.
- Using the previous query example, we could ORDER BY employee_id to see the jobs each employee has held.

```
SELECT hire_date, employee_id, job_id
FROM employees
UNION
SELECT TO_DATE(NULL),employee_id, job_id
FROM job_history
ORDER BY employee_id;
```

SET Operations
ORDER BY

```
SELECT hire_date, employee_id, job_id
FROM employees
UNION
SELECT TO_DATE(NULL),employee_id, job_id
FROM job_history
ORDER BY employee_id;
```

HIRE_DATE	EMPLOYEE_ID	JOB_ID
17/Jun/1987	100	AD_PRES
21/Sep/1989	101	AD_VP
-	101	AC_ACCOUNT
-	101	AC_MGR
13/Jan/1993	102	AD_VP
-	102	IT_PROG
03/Jan/1990	103	IT_PROG
21/May/1991	104	IT_PROG
07/Feb/1999	107	IT_PROG
-	114	ST_CLERK
…	…	…

SET Operations
ORDER BY

- We could improve the readability of the output, by including the start date and end date columns from the job history table, to do this, we would need to match the columns in both queries by adding two more TO_DATE(NULL) columns to the first query.

```
SELECT hire_date, employee_id, TO_DATE(null) start_date,
   TO_DATE(null) end_date, job_id, department_id
FROM   employees
UNION
SELECT TO_DATE(null), employee_id, start_date, end_date, job_id,
   department_id
FROM job_history
ORDER BY employee_id;
```

SET Operations
ORDER BY

HIRE_DATE	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
17/Jun/1987	100	-	-	AD_PRES	90
21/Sep/1989	101	-	-	AD_VP	90
-	101	21/Sep/1989	27/Oct/1993	AC_ACCOUNT	110
-	101	28/Oct/1993	15/Mar/1997	AC_MGR	110
13/Jan/1993	102	-	-	AD_VP	90
-	102	13/Jan/1993	24/Jul/1998	IT_PROG	60
03/Jan/1990	103	-	-	IT_PROG	60
21/May/1991	104	-	-	IT_PROG	60
07/Feb/1999	107	-	-	IT_PROG	60
-	114	24/Mar/1998	31/Dec/1999	ST_CLERK	50
-	122	01/Jan/1999	31/Dec/1999	ST_CLERK	50
16/Nov/1999	124	-	-	ST_MAN	50
17/Oct/1995	141	-	-	ST_CLERK	50
29/Jan/1997	142	-	-	ST_CLERK	50
15/Mar/1998	143	-	-	ST_CLERK	50
…	…	…	…	…	…

Terminology

Key terms used in this lesson included:

- INTERSECT
- MINUS
- SET operators
- TO_CHAR(null) – matching the select list
- UNION
- UNION ALL

Summary

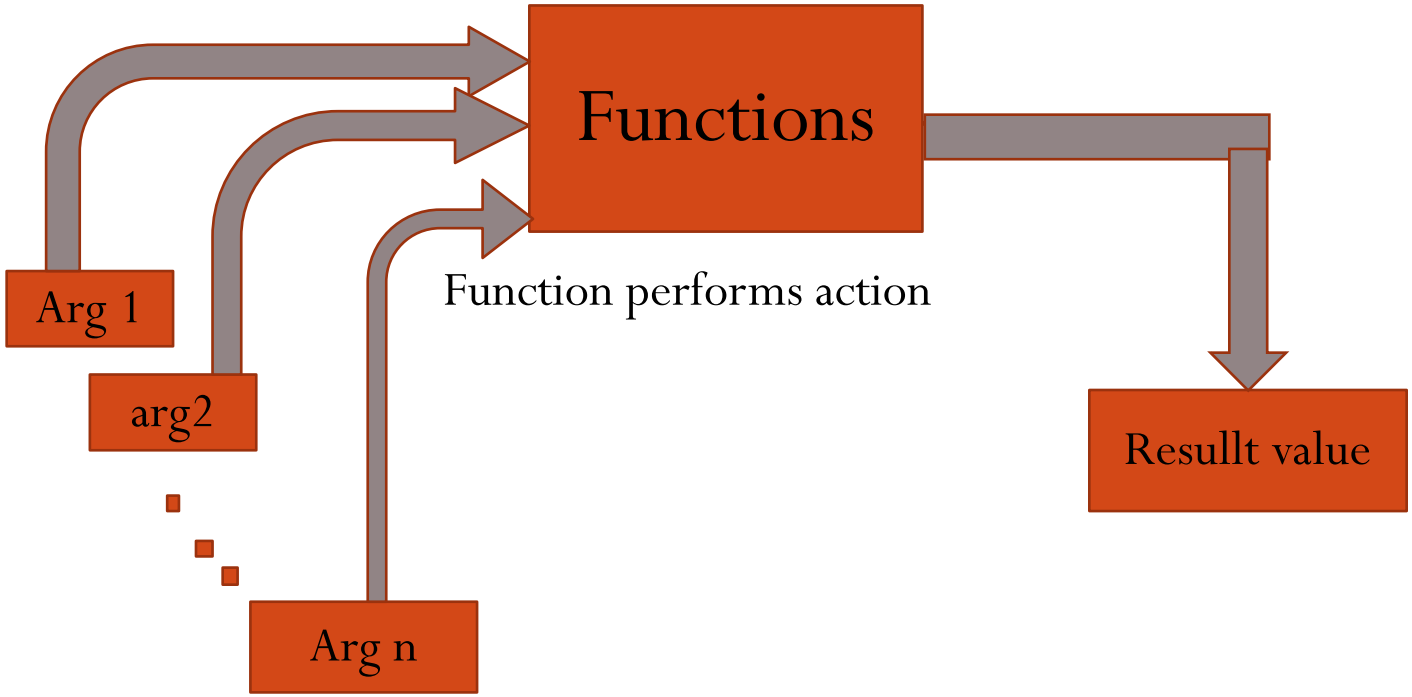
- In this lesson, you should have learned how to:
  - Define and explain the purpose of Set Operators
  - Use a set operator to combine multiple queries into a single query
  - Control the order of rows returned using set operators

SQL FUNCTIONS

Functions

- Functions are very powerful feature of SQL used to manipulate data items .
- SQL functions are built into oracle database and are operated for use in various appropriate SQL statements.
- If you call a SQL function with a null argument, then the SQL function automatically returns null. The only SQL functions that do not necessarily follow this behavior are CONCAT, NVL, REPLACE, and REGEXP_REPLACE.
- Functions are similar to operators in that they manipulate data items and return a result.

SQL FUNCTION

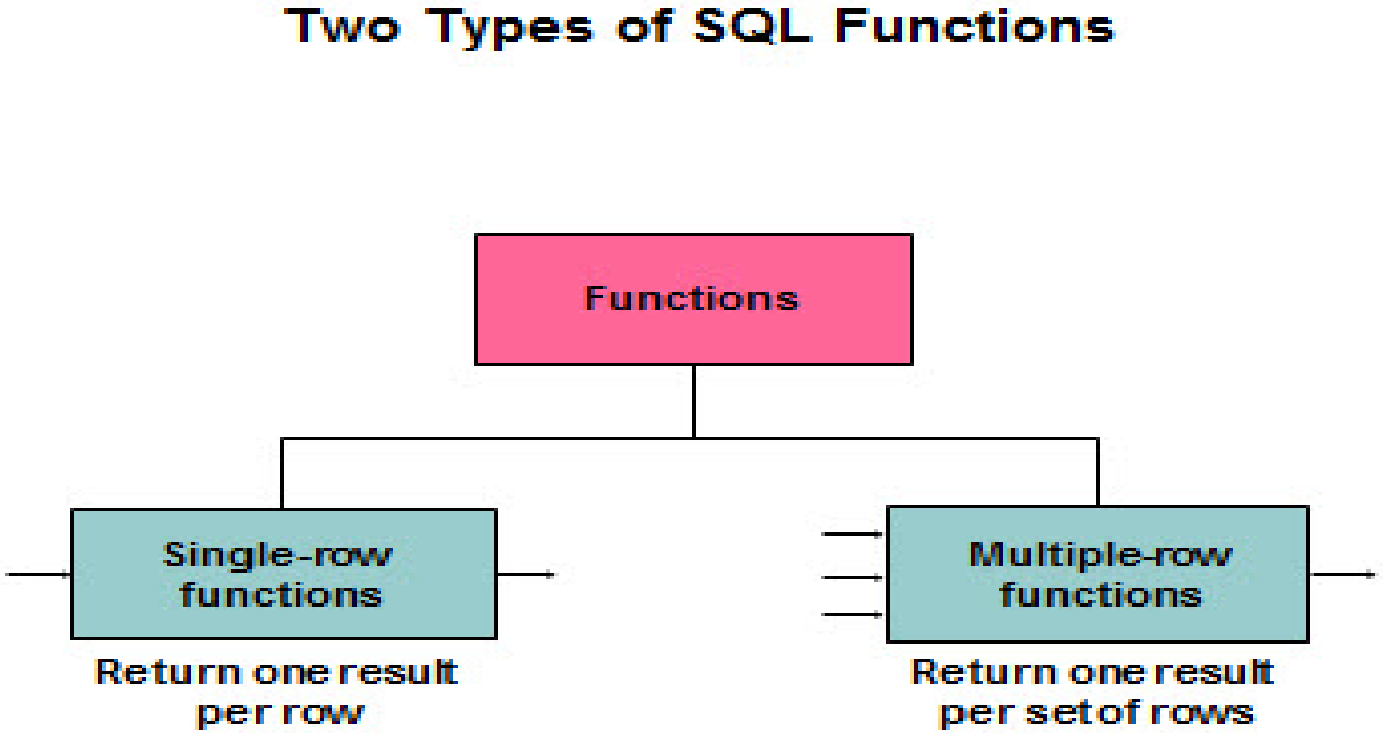


Advantages of function

- Function can be used to perform complex calculations on data.
- Functions can modify individual data items
- Function can very easily manipulate output for groups of rows. Function can manipulate character as well as numeric type of data.
- function can alter date formats for display

TYPES OF FUNCTION

- There are two types of function:
- Single row functions
- Multiple row functions



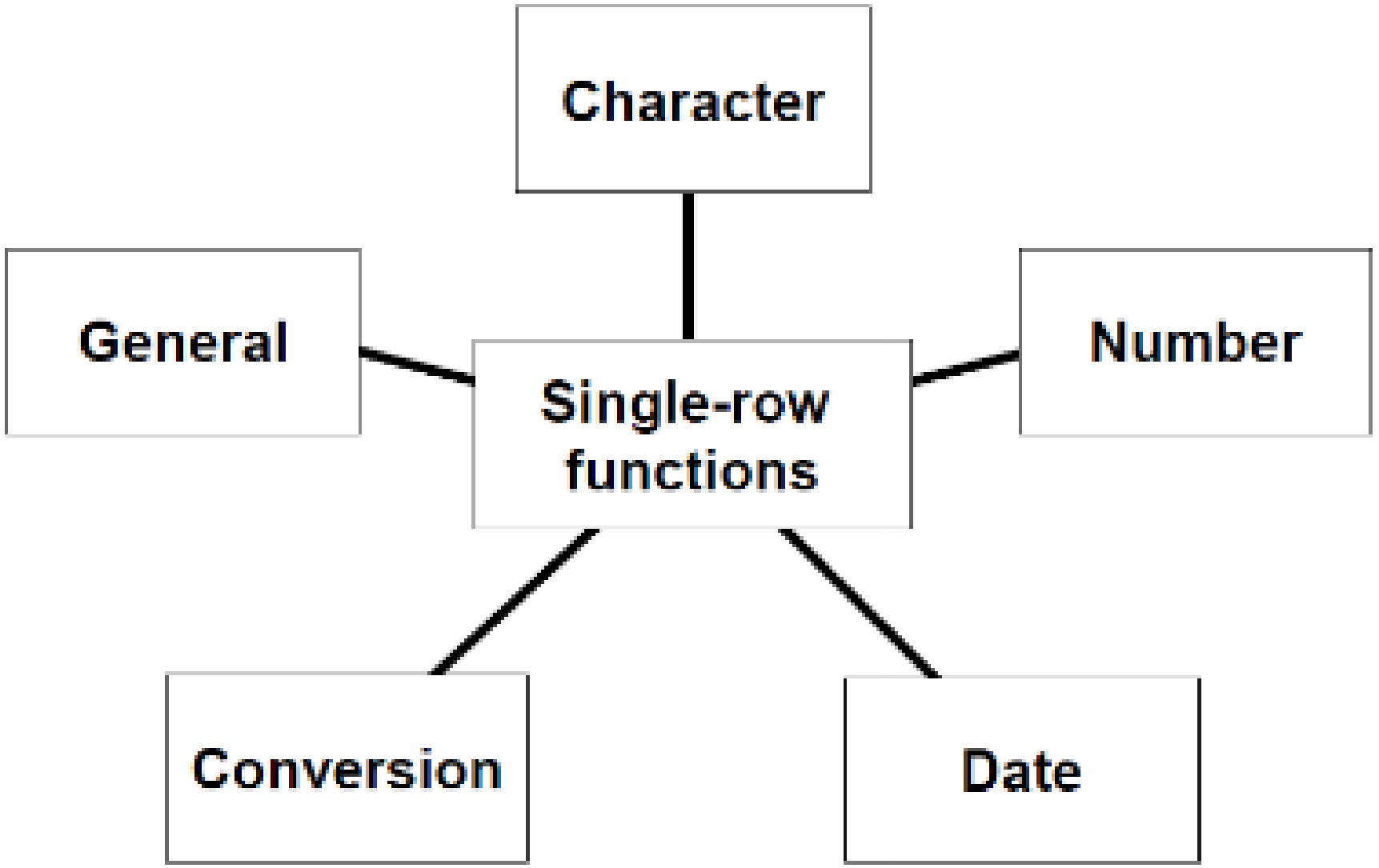
Single row function

- These function operate on single rows only and return one value for ach row, column name or an expression. Single-row functions can be used in SELECT. WHERE and ORDER by clauses.
    - **Syntax of using a single-row function is function_name [(arg1, arg2,…..)]**

- Where, **function_name** is the name of the function. **arg1**,**arg2** is any argument to be used by the function. This can be represented by a user-supplied constant value, variable value, column name or an expression.

Types of single row functions

- There are different types of single row function:
- Character functions
- Number functions/arithmatic functions
- Date functions
- conversion functions
- General functions
- Aggregate functions


Character
General
Single-row functions
Number
Conversion
Date

String/character function

- 1.**LOWER**:- returns char, with all letters in lowercase

  Syntax:-lower(char)

  e.g. select lower('IVAN BAYROSS')"Lower" from dual;

  Output=ivan bayross

2.**INITCAP**:- returns a string with the first letter of each word in upper case.

  Syntax:- initcap(char)

  e.g. select initcap('IVAN BAYROSS')"Title case" from dual;

  Output=Ivan Bayross

- **3.UPPER**:- returns char, with all letters in uppercase.

  syntax:- upper(char)

  e.g. select upper('ivan bayross')"capitalized" from dual;

  Output= IVAN BAYROSS

4.SUBSTR:-returns a portion of characters beginning at character m, and going up to character n. if n is omitted the result returned is up to the last character in the string. The first position of char is 1.

Syntax:- substr(<string>,<start_position>,[<length>])

- Where string is source string
- start_position is the position for extraction. The first position in the string is always 1.
- Length is the number of character is extract.

e.g. select substr(“secure”,3,4) ”Substring” from dual;

Output= cure

5.**ASCII**:-returns the number code that represents the specified character. If more than one character is entered, the function will return the value for the first character and ignore all the characters after the first.

syntax:-ascii(character)

e.g. select ascii('a') "Ascii 1",
ascii('A')"ascii 2", ascii('cure')"ascii " from dual;

ouput= 97 65 99

- **6.COMPOSE**:- return a unicode string. It can be a char, ncahr, nvchar2, clob or nclob.

  Syntax:-compose(<single>)

  Below it is a listing of unistring values that can be combined with other characters in compose function.

unistring value	resulting character
UNISTR('\0300')	grave accent(')
UNISTR('\0301')	acute accent(`)
UNISTR('\0302')	circumflex(^)
UNISTR('\0303')	tilde(~)
UNISTR('\0308')	umlauted(")

- **7.DECOMPOSE**:- accept a string and returns as unicode string.

  Syntax:-decompose(<single>)

8. LENGTH:- returns a length of a word.

Syntax:- length(word)

e.g. select length('sharanam') "length" from dual;

Output= 8

- **9.LTRIM**:- returns characters from the left of char with initial characters removed upto the first character not in set.

  Syntax:-ltrim(char[,set])

  e.g. select ltrim('nisha','n')"ltrim" from dual;

  Output= isha

10. RTRIM:- returns char, with final characters removed after the last character not in set. 'set' is optional, it defaults to spaces.

  Syntax:- rtim(char[,set])

  e.g. select rtrim('sunila','a')"rtrim" from dual;

  Output= sunil

- **11.TRIM**:- remove all specified character either from beginning or the ending of a string.

  Syntax:-

  trim([leading|trailing|both[<trim_charac ter> from]]<string>)

  e.g. select trim(‘ hansel ‘)”trim both side” from dual;

  Output=hansel

  e.g. select trim(leading ‘x’ from ‘xxxhanselxxx’)”remove prefixes” from dual;

  Output= hanselxxx

  e.g. select trim(both ‘x’ from ‘xxxhanselxxx’) from dual;

  Output=hansel

- **12.LPAD**:- returns char1, left-papped to length n with the sequence of character specified in char2.

  Syntax:- lpad(‘char1,n[,char2])

  E.g. select lpad(‘page1’,10,’*’)”lpad” from dual;

  Output=*****page1

13. RPAD:- returns char1, right papped to length n with the character specified in char2.

  Syntax:- rpad(char1,n[,char2])

  e.g. select rpad(ivan,10,’x’)”rpad” from dual;

  Output=ivanxxxxxx

- **14. VSIZE**:- returns the number of bytes in the internal representation of an expression.

  Syntax:- vsize(<expression>)

  e.g. select vsize('sct on the net')"size" from dual;

  Output= 14

15. INSTR:-returns a location of a substring in a string.

Syntax:-

instr(<string1>,<string2>,[<start_position >],[<nth_appearance>])

e.g. select instr('sct on the net','t'), instr('sct on the net','t',1,2) from dual;

Output= 8 14

NUMERIC FUNCTIONS.....

- 1. **ABS**:- returns the absolute value of 'n'.

  syntax:- ABS(-15)

  e.g. Select ABS(-15) "absolute" from
  dual;

- 2.**POWER**:- returns m raised to the nth power. n must be an integer else an error is returned.

  syntax:-power(m,n)

  e.g. Select power(3,2)"raised" from
  dual;

- 3.**Round**:-returns n, rounded to m places to the right of the decimal point. If m is omitted, n is rounded to 0 places, m can be negative to round off digits to the left of the decimal point. m must be an integer

  syntax:-round(n,[m])

  e.g. select round(15.91,1) from dual;

  output=15.2

4.**SQRT**:- returns square root of n.

  syntax:-sqrt(n)

  e.g. select sqrt(25) from dual;

  output=5

- **5.EXP**:-returns e raised t the nth power where e=2.71828183

  syntax:- exp(n)

  E.g. select exp(5) from dual;

  Output=148.413159

- **6.EXTRACT**:-returns a value extracted from a date or an integer value. A date can be used only to extract year, month and day, while a timestamp with a time zone data type can be used only to extract timezone_hour and timezone_minute.

  E.g. select extract(year from date '2004-07-02')"year", extract(month from sysdate)"month" from dual;

  Output=2004 7

- **7. GREATEST :-** returns a greatest value in a list of expressions.

  Syntax:-greatest(expr1,expr2,expr3…expr n)

  e.g.:- select greatest(4,5,17)”num”,
  greatest(‘4’,’5’,’17’)”text” from dual;

  output= 17 5

- **8.LEAST**:- returns the least value in a list of expressions.

  Syntax:- least(expr1,expr2,……,exprn);

  e.g. select least(4,5,17)”num”,
  least(‘4’,’5’,’17’)”text” from dual;

  Output= 4 17

- **9.MOD** :-returns the remainder of a first number divided by second number passed a parameter. If the second number is zero the result of the same as the first number

  Syntax:-mod(m,n)

  e.g. select mod(15,7)"mod1",
  mod(15.7,7)"mod2" from dual;

  Output= 1 1.7

- **10.TRUNC**:- returns a number truncated to a certain no. of decimal places. The decimal place value is must be an integer.

  Syntax:- trunc(no,[decimal_places])

  e.g. select trunc(125.815,1)"trunc1",
  trunc(125.815,-2)"trunc2" from dual;

  Output= 125.8 100

11. FLOOR:- return a largest integer value that is equal to less than a number.

Syntax:-floor(n)

e.g. select floor(24.8)"flr1", floor(13.15)"flr2" from dual;

Output=24 13

12.CEIL:-return the smallest integer value that is greater than or equal to a number.

Syntax:-ceil(n)

e.g. select ceil(24.8)"ceil", ceil(13.15)"ceil2" from dual;

Output= 25 14

CONVERSION FUNCTIOSNS

- These are functions that help us to convert a value in one form to another form. For example: a null value into an actual value, or a value from one datatype to another datatype . Few of the conversion functions available in oracle are:
- **TO CHAR(d,f)**

This function converts the date 'd' to character format 'f'.

Example:

SELECT SYSDATE,TO_CHAR(SYSDATE,'DAY') FROM DUAL;

OUTPUT:

SYSDATE	*TO _CHAR(S*
03-SEP-13	*TUESDAY*

- **TO_DATE(char,f)**

This function converts the character string representing date into a date format according to 'f' format specified. If no format is specified, then the default format is **DD-MON-YY.**

Example:

SELECT SYSDATE,TO_DATE('JAN2007','MONYYYY') FROM DUAL;

Output:

```
SYSDATE      TO_DATE(
-----------  -------------------------
03-SEP-13        01-JAN-07
```

- **NVL(col,value)**

This function helps in substituting a value in place of a null value. The data type of the value to substitute must match with the col data type.

Example:

Select nvl(null,101) from dual;

Output:

```
Nul(null.101)
-------------------
            101
```

- **DECODE(a,b,c,d,e,default_value)**
- This function substitutes on a value-by value basis, it actually dows an '**if-then –else**' test. It checks the value of 'a', if a=b, then returns **'c'**. If a=d, then results 'e'. Else, returns default value.

Example:

SELECT ENAME, JOB,

DECODE(JOB,'CLERK',EXECUTIVE','MANAGER','GM',' CASHIER') FROM EMP;

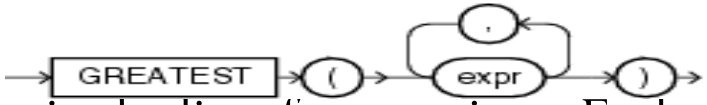
Output:

ENAME	*JOB*	*DECODE(JO*
SMITH	*CLERK*	*EXECUTIVE*
ALLEN	*SALESMAN*	*CASHIER*
WARD	*SALESMAN*	*CASHIER*
JONES	*MANAGER*	*GM*

General functions

- The general comparison functions determine the greatest and least value from a set of values. Some general functions also help to find the detail of current database user. Few of the general functions available in oracle are:

Greatest(exp1,exp2,exp3….)



- This function returns the greatest value in the list of expressions. Each expression is implicitly converted to the type of expression (exp1) before the comparison are made .
- If the first expression is numeric, then the oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that data type before the comparison , and return that data type.
- If the first expression(exp1) is not numeric, then each expression after the first is implicitly converted to the data type of the first expression before the comparison.

Example:

SELECT GREATEST(33,55,66) FROM DUAL;

OUTPUT:

GREATEST(33,55,66)

66

EXAMPLE:

SELECT GREATEST (‘R’,’A’,’Z’) FROM DUAL;

OUTPUT:

G

Z

EXAMPLE :

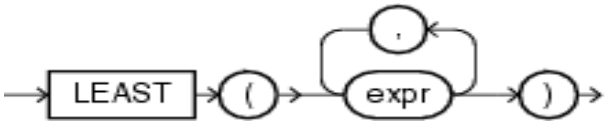
GREATEST(‘HARD’,’HARRY’,’HAROLD’) FROM DUAL;

OUTPUT:

GREAT

HARRY

- **Least(exp1,exp2,exp3…)**



This function returns the least value in the list of expressions. LEAST function behaves same like Greatest , in which all expressions are implicitly converted to the data type of the first.

EXAMPLE:

Select least(44,22,7) from dual;

Output:

Least(44,22,7)

7

UID

THIS FUNCTION RETURNS AN INTEGER THAT UNIQUELY IDENTIFIES THE CURRENT DATABASE USER. UID TAKES NO ARGUMENTS.

EXAMPLE:

SELECT UID FROM DUAL;

OUTPUT:

UID

57

- **USER**

This function returns a vaharchar2 value containing the name of the current oracle user. User function takes no arguments.

EXAMPLE:

SELECT USER FROM DUAL;

OUTPUT:

USER

SCOTT

AGGREGATE FUNCTIONS.....

- 1.**AVG** :- returns the average value

  syntax:- Select avg(sal) from emp;

- 2.**MIN** :- return the minimum value of expr.

  syntax :-select min(sal) from emp;

- 3.**COUNT** :- returns the no. of rows where expr. Is not null

  syntax:-select count(acct_no) from acct_mstr;

- 4.**COUNT(*)** :- Returns the no. of rows in a table including duplicates and those with null.

  syntax:- select count(*)"no of records"
  from acct_mstr;

- 5.**MAX**:- Returns the minimum value of expr.

  syntax:-select max(curbal) from
  acct_mstr;

- 6.**SUM**:-Returns the sum of the value of 'n'

  syntax:-select sum(curbal) from acct_mstr;

DATE FUNCTIONS

- Oracle database stores date in an internal numeric format, representing the century ,Year, month, day hours, minutes, and seconds. The default date display format is DD_MON_YY.
- Date function operates on oracle dates. These are the function that takes values of DATE datatype as input and return values of date datatype as output, except for the MONTHS_BETWEEN function, which returns a number as output. Few date functions are as given below.

- SYSDATE

SYSDATE is a pseudo-column that returns the system's current date and time of type DATE. The SYSDATE can be used just as any other column name. it takes no arguments. When used in distributed SQL statements, SYSDATE returns the date and time of the local database.

Example:

SELECT SYSDATE FROM DUAL;

output: O3-SEP-13

- *ADD_MONTH(d,n)*

This function adds or subtract months to or from date, it returns a date as result.

Example:

SELECT SYSDATE, ADD_MONTHS(SYSDATE,4) FROM DUAL;

OUTPUT:

SYSDATE ADD_MONTHS

--- ---

03-APR-13 03-AUG-13

- **MONTHS_BETWEENd1,d2,**

This function returns the number of months between two dates, d1and d2. if d1 is later than d2, then the result is positive. If d1 is earlier than d2, then the result is negative. The output will be a number.

Example

SELECT MAONTHS_BETWEEN("25-DEC-81',25-DEC-79') AS DATE1, MONTHS_BETWEEN('25-DEC-79','25-DEC-81') AS DATE2 FROM DUAL;

OUTPUT:

DATE1 DATE2

--------- ----------

24 **-24**

- **NEXT_DAY(DATE,DAY)**

THIS FUNCTION RETURNS THE DATE OF NEXT SPECIFIED DAY OF THE WEEK AFTER THE 'DATE'.

EXAMPLE

SELECT SYSDATE, NEXT_DAY(SYSDATE,'FRIDAY) FROM DUAL;

OUTPUT:

SYSDATE NEXT_Day(

------------ --------------

03-SEP-13 06-SEP-13

- **LAST_DAY(d)**

This function returns the date of the last day of the month specified. The result will be a date.

Example:

SELECT SYSDATE, LAST_DAY(SYSDATE) FROM DUAL;

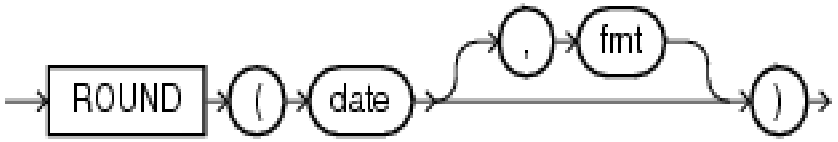
OUTPUT:

SYSDATE LAST_DAY(

--------- ----- ----------

03-SEP-13 30-SEP-13

- **ROUND(d[,format])**



This function rounds the date d to the unit specified by format. If format is not specified, is default to 'DD' , which rounds d to the nearest day.

Example:

SELECT SYSDATE, ROUND(SYSDATE,'MM') AS "NEAREST MONTH" FROM DUAL;

OUTPUT:

SYSDATE NEAREST M

--------- ---------

03-SEP-13 01-SEP-13

- **TRUNC(d[,formt])**

This function returns the date d truncated to the unit specified by format. If format is omitted, then it defaults to 'DD', which truncates d to the nearest day.

Example:

SELECT SYSDATE,TRUNC(SYSDATE,'YEAR') AS "FIRST DAY" FROM DUAL;

OUTPUT:

SYSDATE	FIRST DAY
03-SEP-13	01-JAN-13

SPECIAL DATE FORMAT USING TO_CHAR FUNCTION

- Sometimes the date value is required to be displayed in special format for e.g. instead of 03-jan-81, displays the date as 3rd of January 1981. for this oracle provides special attributes, which can be used in the format specified with the to char and to date functions. The significance and use of these characters are explained in the examples…….

Use of th in the to_char() function

- DDTH places TH, RD, ND for the date like 2nd, 3rd, 8th etc……

  e.g. select cust_no,To_char(dob_inc,'ddth-mon-yy') "DOB_INC" from cust_master;

  OUTPUT====

CUST_NO	DOB_INC
C1	25TH-JUN-52
C2	29TH-OCT-82
C3	28TH-OCT-75
C4	02ND-APR-79
….	……

USE OF SP IN THE TO_CHAR() FUNCTION

- indicates that the date(dd) must be displayed by spelling such as one, twelve.

e.g. select
cust_no,to_char(dob_inc,'DDSP')"DOB_DDSP "
from cust_master;

Output==========

CUST_NO	DOB_DDSP
C1	TWENTY-FIVE
C2	TWENTY -NINE
C3	TWENTY-EIGHT
C4	TWO
….	…….

USE OF SPTH IN THE TO_CHAR FUNCTION

- Displays the date (dd) with th added to the spelling like fourteenth, twelfth.

e.g. select cust_no,to_char(dob_inc,'DDSPTH')"DOB_DDSPTH" from cust_master;

Output==========

CUST_NO	DOB_DDSPTH
C1	TWENTY-FIFTH
C2	TWENTY -NINTH
C3	TWENTY-EIGHTH
C4	SIXTH
….	……

SQL Joins

Types of Joins

- Inner Join
  - Natural Join
- Left (Outer) Join
- Right (Outer) Join
- (Full) Outer Join
- Left (Outer) Join Excluding Inner Join
- Right (Outer) Join Excluding Inner Join
- (Full) Outer Join Excluding Inner Join
- Cross Join
- Equi-Join

Sample Tables

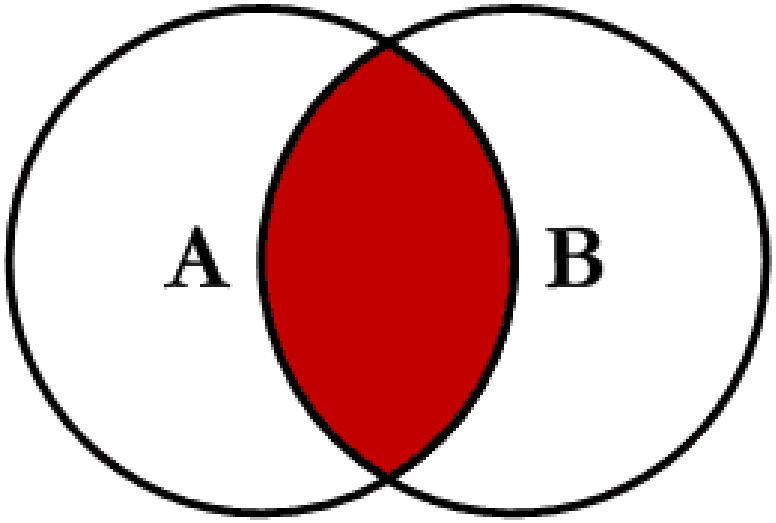
Table A

PK	Value
1	FOX
2	COP
3	TAXI
6	WASHINGTON
7	DELL
5	ARIZONA
4	LINCOLN
10	LUCENT

Table B

PK	Value
1	TROT
2	CAR
3	CAB
6	MONUMENT
7	PC
8	MICROSOFT
9	APPLE
11	SCOTCH

Inner Join



- **Inner join** produces only the set of records that match in both Table A and Table B
- Most commonly used, best understood join

Inner Join

Table A Value	PK	Table B PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC

SELECT * FROM TableA **INNER JOIN** TableB ON
TableA.PK = TableB.PK

- This is the same as doing
  SELECT * FROM TableA, TableB **WHERE** TableA.PK = TableB.PK

Inner Join (continued)

- Inner Joins do not have to use equality to join the fields
- Can use <, >, <>

Inner Join (continued)

```
SELECT *
FROM
TableA INNER
JOIN TableB
ON
TableA.PK >
TableB.PK
```

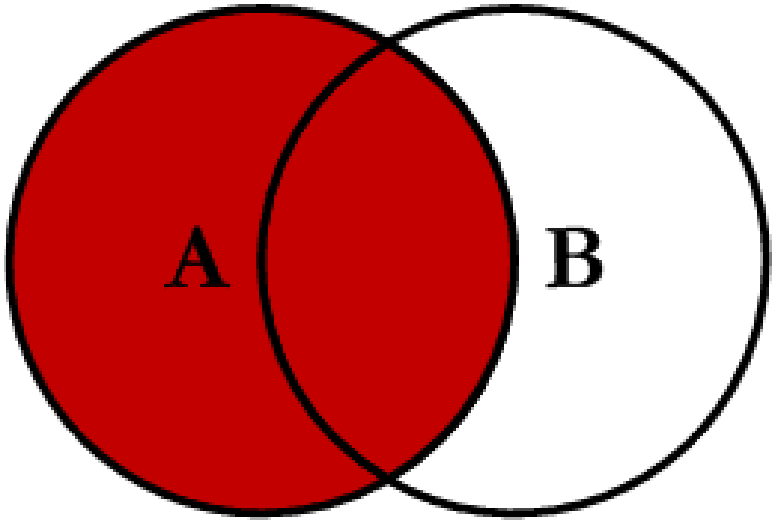
Table A PK	Value	Table B PK	Value
2	COP	1	TROT
3	TAXI	1	TROT
3	TAXI	2	CAR
4	LINCOLN	1	TROT
4	LINCOLN	2	CAR
4	LINCOLN	3	CAB
5	ARIZONA	1	TROT
5	ARIZONA	2	CAR
5	ARIZONA	3	CAB
…	More…	Rows…	

Inner Join/Natural Join

- A NATURAL join is just an inner equi-join where the join is implicitly created using **any** matching columns between the two tables
- Example:
  - SELECT * FROM TableA **NATURAL JOIN** TableB
  - Same results as inner equi-join?
  - Which columns match?

Left Outer Join

- Left outer join produces a complete set of records from Table A, with the matching records (where available) in Table B. If there is no match, the right side will contain null.



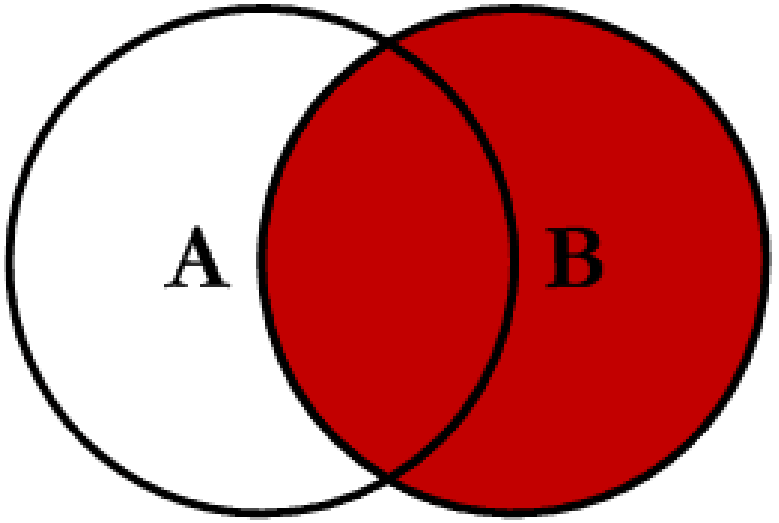
Left Outer Join

Table A Value	PK	Table B PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
LINCOLN	4	NULL	NULL
ARIZONA	5	NULL	NULL
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC
LUCENT	10	NULL	NULL

- SELECT * FROM TableA LEFT OUTER JOIN TableB ON TableA.PK = TableB.PK

Right Outer Join

- Right outer join produces a complete set of records from Table B, with the matching records (where available) in Table A. If there is no match, the left side will contain null.



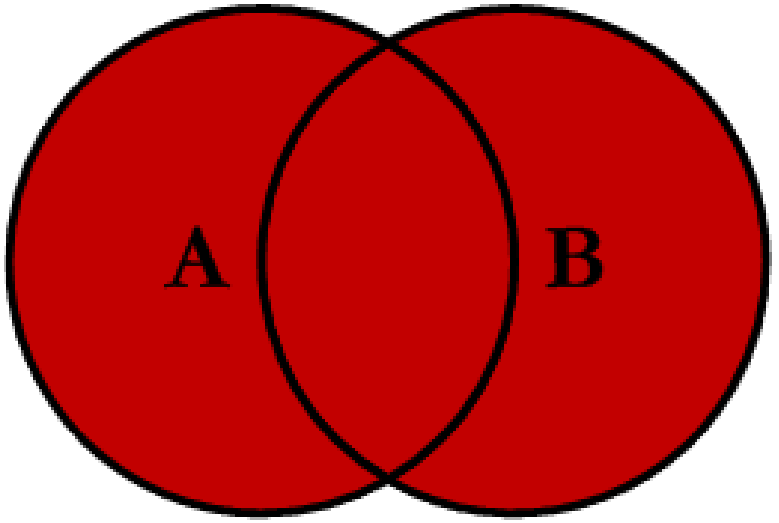
Right Outer Join

Table A Value	PK	Table B PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC
NULL	NULL	8	MICROSOFT
NULL	NULL	9	APPLE
NULL	NULL	11	SCOTCH

- SELECT * FROM TableA RIGHT OUTER JOIN TableB ON TableA.PK = TableB.PK

Full Outer Join

- Full outer join produces the set of all records in Table A and Table B, with matching records from both sides where available. If there is no match, the missing side will contain null.



Full Outer Join

TableA Value	PK	TableB PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
LINCOLN	4	NULL	NULL
ARIZONA	5	NULL	NULL
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC
LUCENT	10	NULL	NULL
NULL	NULL	8	MICROSOFT
NULL	NULL	9	APPLE
NULL	NULL	11	SCOTCH

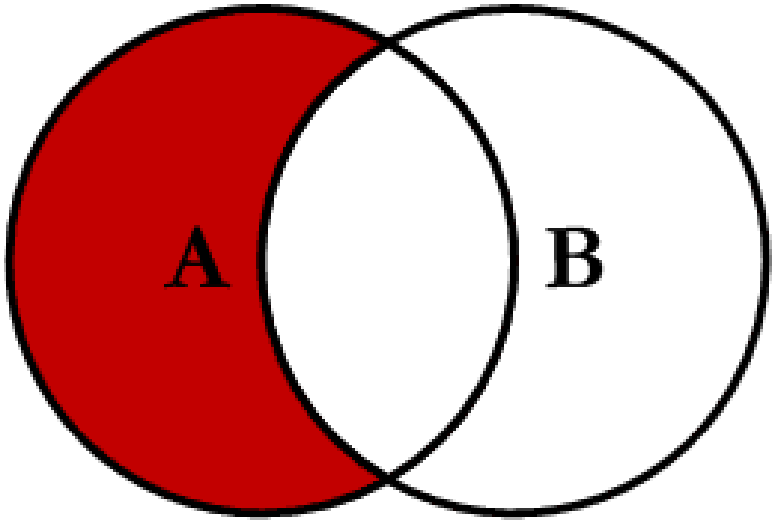
- SELECT * FROM TableA FULL OUTER JOIN TableB ON TableA.PK = TableB.PK

Full Outer Join in MySQL

- MySQL doesn't have FULL OUTER JOIN
- Simulate using UNION, LEFT and RIGHT JOINs
- SELECT * FROM TableA LEFT JOIN TableB ON TableA.PK = TableB.PK
  UNION
  SELECT * FROM TableA RIGHT JOIN TableB
  ON TableA.PK = TableB.PK

Left Join Excluding Inner Join

- This query will return all of the records in the left table (table A)that do not match any records in the right table (table B).



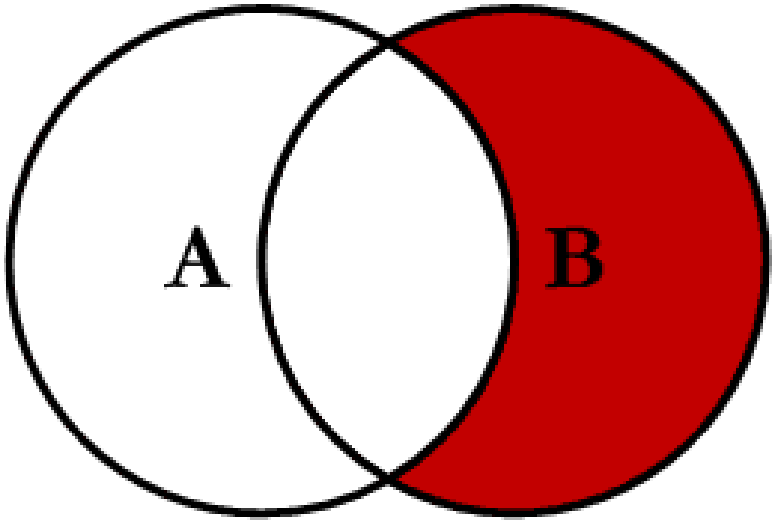
Left Join Excluding Inner Join

Table A Value	PK	Table B PK	Value
LINCOLN	4	NULL	NULL
ARIZONA	5	NULL	NULL
LUCENT	10	NULL	NULL

- SELECT * FROM TableA LEFT JOIN TableB ON TableA.PK = TableB.PK WHERE TableB.PK IS NULL
- Perform left outer join, then exclude the records we don't want from the right side via a where clause.

Right Join Excluding Inner Join

- This query will return all of the records in the right table (table B) that do not match any records in the left table (table A).



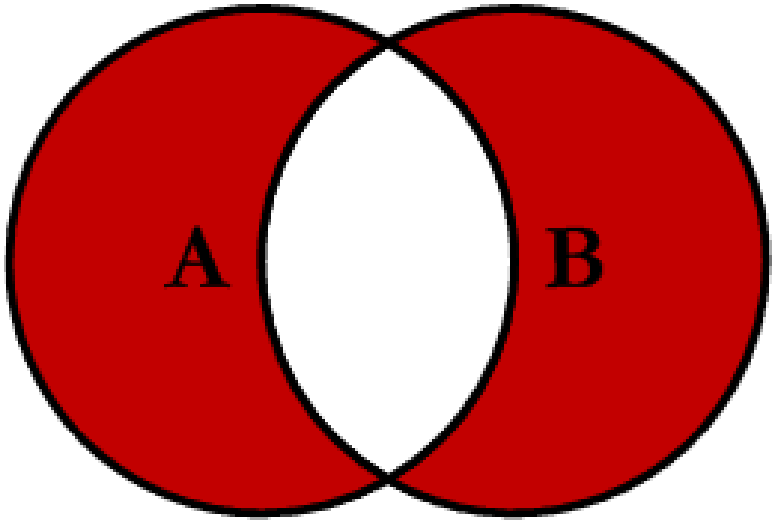
Right Join Excluding Inner Join

Table A Value	PK	Table B PK	Value
NULL	NULL	8	MICROSOFT
NULL	NULL	9	APPLE
NULL	NULL	11	SCOTCH

- SELECT * FROM TableA RIGHT JOIN TableB ON TableA.PK = TableB.PK WHERE TableA.PK IS NULL
- Perform right outer join, then exclude the records we don't want from the left side via a where clause.

Full Outer Excluding Inner Join

- This query will return all of the records in Table A and Table B that do not have a matching record in the other table.
- (If you find a useful application, let me know! ☺)



Full Outer Excluding Inner Join

Table A Value	PK	Table B PK	Value
NULL	NULL	8	MICROSOFT
NULL	NULL	9	APPLE
NULL	NULL	11	SCOTCH
LINCOLN	4	NULL	NULL
ARIZONA	5	NULL	NULL
LUCENT	10	NULL	NULL

- SELECT * FROM TableA FULL OUTER JOIN TableB ON TableA.PK = TableB.PK WHERE TableA.PK IS NULL OR TableB.PK IS NULL

How Can We Do This in MySQL?

- MySQL doesn't have FULL OUTER JOIN
- Simulate using UNION, LEFT and RIGHT JOINs
- SELECT * FROM TableA LEFT JOIN TableB ON TableA.PK = TableB.PK WHERE TableB.PK IS NULL UNION SELECT * FROM TableA RIGHT JOIN TableB ON TableA.PK = TableB.PK WHERE TABLEA.PK IS NULL

Cross Join

- A cross join is a Cartesian Product join – it is every record in Table A combined with every record in Table B.
- It gives the same results as not using a WHERE clause when querying two tables in MySQL
- SELECT * from TableA CROSS JOIN TableB
- SELECT * from TableA, TableB