SQL :

Data: Any thing which can be typed …transmittable and storable in computer system.

Information : organized meaning ful data is called information.

Database : A **database** is an organized collection of [data](https://en.wikipedia.org/wiki/Data_(computing)).[[1]](https://en.wikipedia.org/wiki/Database#cite_note-1) It is the collection of [schemas](https://en.wikipedia.org/wiki/Database_schema), [tables](https://en.wikipedia.org/wiki/Table_(database)), [queries](https://en.wikipedia.org/wiki/Query_language), reports, [views](https://en.wikipedia.org/wiki/View_(SQL)), and other objects.

Database Management System : A ***d***ata***b***ase***m***anagement***s***ystem (DBMS) is a collection of [programs](http://www.webopedia.com/TERM/P/program.html) that enables you to [store](http://www.webopedia.com/TERM/S/store.html), modify, and extract information from a [database](http://www.webopedia.com/TERM/D/database.html). There are many different types of database management systems, ranging from small [systems](http://www.webopedia.com/TERM/S/system.html) that [run](http://www.webopedia.com/TERM/R/run.html) on [personal computers](http://www.webopedia.com/TERM/P/personal_computer.html) to huge systems that run on [mainframes](http://www.webopedia.com/TERM/M/mainframe.html).

A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database.

System establishing the relation between diff tables, entities…etc.

DBA shows us relationships in ER Diagrams..

Types of relations :

One –One: ordered – product id : unique ids : ssn, DL

One – Many: order – multiple products

Many – One: multiple customers – buy single item

Many – Many: many people – many products

To download sql developer:

Oracle.com---Register with email id--- download sql developer---extract into system---double click on sql developer )tool which helps to execute the queries)

First need to exstablish the connection in realtime scenarios :

Give any connection name but username and pwd shoulb be given correct credentioals which are shared to us.n

Host name : if it’s in local system then we give the local address or else hostname which is shared to us by the team members.

Port number : default port numbers will be there.

Once we establish the connection we can write the sql commands….similarly we have tool called Toad for that also we need to establish the connections using credentials.

These are open source tools.

To practice sql we can use Livesql.com.

Schema : set of tables together are organized at one place.

Orcle has different types of statements :

DDL: Data Definition languages

DML: Data Manipulation Languages

DCL: Data Control Languages

TCL: Transaction Control Languages

**DDL**

**Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples: DDL statements can’t be rolled back.

* CREATE - to create objects in the database
* ALTER - alters the structure of the database
* DROP - delete objects from the database, delete entire table including the structure.
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed but structure remains the same.
* COMMENT - add comments to the data dictionary
* RENAME - rename an object

**For drop and truncate we cant give conditions and it cant be rolled back.it delete the all the data from tables.**

**DML**

* DELETE - deletes all records from a table, the space for the records remain

**Data Manipulation Language** (DML) statements are used for managing data within schema objects. Some examples: DML statements can be rolled back.

* SELECT - retrieve data from the a database, getting the data
* 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
* MERGE - UPSERT operation (insert or update)
* CALL - call a PL/SQL or Java subprogram
* EXPLAIN PLAN - explain access path to data
* LOCK TABLE - control concurrency

**DCL**

**Data Control Language** (DCL) statements. Some examples:

* GRANT - gives user's access privileges to database
* REVOKE - withdraw access privileges given with the GRANT command

**TCL**

**Transaction Control** (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

* 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 isolation level and what rollback segment to use

**Distinct**: to get the unique details instead of duplicate data we can use the keyword “distinct” in select statement.

Distinct can be used on multiple columns with where condition.

select distinct salary,first\_name from hr.employees where salary=3000 order by salary desc;

\*: for all the records

**Arithmetic expressions:** +, -, \*, /

Ex. Select \* from salary+100; --performing arithmetic operation on particular column.

**Sorting**: keyword “Order by” is used for sorting the data. Default the data is sorted in ascending order. For descending need to use the keyword “desc” along with “order by”.

Ex.1)Select salary from hr. Employees order by desc;

Ex.2) SELECT last\_name, job\_id, department\_id, hire\_date FROM employees ORDER BY hire\_date;

Order by clause can be used with multiple columns, and always orber by is used at the end of the select statement.

Ex.3) SELECT last\_name, department\_id, salary FROM employees ORDER BY department\_id, salary DESC;

**Alias**: blanks space or keyword “as” is used for alias name. if there is a space in alias name then we need to mention them in quotes “”.

Ex. Select \* from hr. Employees as emp;

Select salary sal from hr. Employees;

**NULL Value in SQL**: whenever there is no data(blank space) is given in the columns they default considered as null value. Null doesn’t mean as zero. We can check with “is null” command to check for the null values in the columns

**Is nul**l: select \* from hr. Employees where salary is null;

**Is not null**: select \* from hr. employees where salary is not null;

Null doesn’t mean as zero or blank space. it’s just null empty field. initially sometimes we don’t have the defined data for some columns then we just create the field and data is updated later.

A NULL is a value that is unavailable, unassigned, unknown or inapplicable but not the same as blank space or zero.

All oracle keywords are not case sensitive but the data is case sensitive.

Any arithmetic operation performed on null value result is NULL.

When we don’t know about the case of the data we can use the help of oracle character functions {upper, lower} .

Ex. Select \* from hr. employees where upper(first name)=’john’;

Data can be given in single quotes ‘ hello ‘.

**Concatination Operator:** (pipe symbol ||):A concatenation operator: Links columns or character strings to other columns. Is represented by two vertical bars (||).Creates a resultant column that is a character expression. We can give alias names for the new concatenated column.

Ex. Select first\_name || last\_name as “full name” from hr. Employees;--- here in alias name we have space hence we mentioned in “”.

Ex. Select first\_name || q’[salary’s]’ || salary as “saldisplay” from hr. employees;----- “salary is” not possible hence we wrote as q’[salary’s]’ –syntax.

**Comparison Operators**: <, >, =, <>(not operator),<=, >=

Between and AND operator: it gives the between the first and last values…

Ex. Select \* from hr.employees where salary between 10000 and 5000;

IN: operator is used if we want to check for set of values .

Ex. Select \* from hr.employees where employee\_id in(100,102,107);

AND: if we want check for two conditions “AND” can be used.

Ex. Select \* from hr.employees where salary=10000 and last\_name=”john”;

OR: for either of conditions

Ex. Select \* from hr.employees where salary=10000 or first\_name=”jenny”;

If we want to use both “and” and “OR” operator together then we need to use paramthesis to separate the conditions. Ex. Select \* from hr.employees where (salary=1000 and empid=1) or (last\_name=”hari”);

**Like**: is for particular selection compare strings starts with ends with any type of comparisons..give the provision. Like ‘%s’..used with Strings.

Ex. Select \* from hr.employees where last\_name like ‘k%’;

Literal Character Strings:

A literal is a character, a number, or a date that is included in the SELECT statement. Date and character literal values must be enclosed by single quotation marks.– Each character string is output once for each row returned.

If we wont give percentile symbols then it works as =.

**Comparison Conditions**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| = | Equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| <> | Not equal to |
| BETWEEN...AND... | Between two variables(inclusive) |
| IN(set) | Match any of a list of values |
| LIKE | Match a character pattern |
| IS NULL | Is a null value |

Ex. SELECT last\_name, salary FROM employees WHERE salary BETWEEN 2500 AND 3500 ;

Ex. SELECT employee\_id, last\_name, salary, manager\_id FROM employees WHERE manager\_id IN (100, 101, 201) ;

Ex. SELECT last\_name FROM employees WHERE last\_name LIKE '\_o%' ;

Ex. SELECT last\_name, manager\_id FROM employees WHERE manager\_id IS NULL ;

**Logical Operators:**

**AND: requires both conditions to be true**

Ex. SELECT employee\_id, last\_name, job\_id, salary FROM employees WHERE salary >=10000 AND job\_id LIKE '%MAN%';

**Or: requires either the condition to be true**

SELECT employee\_id, last\_name, job\_id, salary FROM employees WHERE salary >= 10000 OR job\_id LIKE '%MAN%';

**NOT:**

SELECT last\_name, job\_id FROM employees WHERE job\_id NOT IN P'IT\_PROG', 'ST\_CLERK', 'SA\_REP');

SQL Functions:

1. Single row functions:for each row one output. Ex. Select lower(firstname) from hr.employees;
2. Multi row functions : single output for set of rows. Performing on multiple rows and getting the single row output. Ex. Select max(salary) from hr.employees;

Single-row functions: for every row we get one output

– Manipulate data items

– Accept arguments and return one value

– Act on each row that is returned

– Return one result per row

– May modify the data type

– Can be nested

– Accept arguments that can be a column or an expression

**Character Functions:**

1. **Case Manupulation Functions**
2. Upper : UPPER('SQL Course')—SQL COURSE
3. Lower: LOWER('SQL Course')---sql course
4. Initcap: INITCAP('SQL course)—Sql Course

Ex. SELECT employee\_id, last\_name, department\_id FROM employees WHERE LOWER(last\_name) = 'higgins';

1. **Character Manupulation functions**
   1. Comcat: CONCAT('Hello', 'World')—HelloWorld--- concatenates two strings
   2. Substr: SUBSTR('HelloWorld',1,5)---Hello – gives the sub string based on the given range of index values
   3. Length: LENGTH('HelloWorld')—10 – gives the length of the string including the spaces
   4. Instr: INSTR('HelloWorld', 'W')—6 index of first occurance of character
   5. Lpad : LPAD(salary,10,'\*')--- \*\*\*\*\*24000 – it appends \* to the left side to make the length as 10
   6. Rpad: RPAD(salary, 10, '\*') ---24000\*\*\*\*\*---appends \* to right side.
   7. Trim: TRIM('H' FROM 'HelloWorld') ---elloWorld: it trims the string at the beginning and end.. characters or spaces but it wont trim spaces between the strings.
   8. Replace: REPLACE('JACK and JUE','J','BL')--BLACK and BLUE

Ex. SELECT employee\_id, CONCAT(first\_name, last\_name) NAME, job\_id, LENGTH (last\_name), INSTR(last\_name, 'a') "Contains 'a'?" FROM employees WHERE SUBSTR(job\_id, 4) = 'REP';

**Dual**: is a table which is already present in oracle with only one column called dummy with varchar2 datatype.

Ex. Select sysdate from dual;

**Data Types in Oracle**: something which specifies the type of data. Most frequently used data types are

1. **Varchar**: string representation—can differentiate between null and space
2. **Varchar2**: is also represents strings but can not differentiate between space and null.
3. **Number**: interger,numbers
4. **Date**: date format default format is dd-mon-rr
5. **Decimal**
6. **Timestamp**: gives date along with time

Etc.

**Number based functions:**

ROUND: Rounds value to specified decimal

TRUNC: Truncates value to specified decimal

MOD: Returns

1. **Round**: ROUND(45.926, 2) --- 45.93 – it rounds the value depending upon the number given.it will be rounded.
2. **Trunk**: just goes to the original position ..truncate the places placevalue remains the same. TRUNC(45.926, 2) ---45.92
3. **MOD**: modulas operator gives the reminder: MOD(1600, 300) ---100

Mangodb is no sql database.

Ex. SELECT ROUND(45.923,2), ROUND(45.923,0),ROUND(45.923,-1) FROM DUAL;

SELECT TRUNC(45.923,2), TRUNC(45.923),TRUNC(45.923,-1) FROM DUAL;

SELECT last\_name, salary, MOD(salary, 5000) FROM employees WHERE job\_id = 'SA\_REP';

**Date Functions**:

select \* from hr.employees where hire\_date > '01/jan/2005';

select sysdate from dual;

select **next\_day**(sysdate,'wednesday') from dual;---

select **add\_months**(sysdate,6) from dual;

select **round**(sysdate,'month') from dual;

select round(to\_date('3-feb-2017'),'month') from dual;

to\_date: converts the date to string

**Conversion** **Functions**:

1. Implicit data type conversions
2. Explicit data type Conversions: explicitly we want to convert the the data types.

To\_char: takes date as the input formart and converts the date format to char date format.

Ex. SELECT last\_name, TO\_CHAR(hire\_date, 'DD Month YYYY') AS HIREDATE FROM employees;

To\_char with numbers:

SELECT TO\_CHAR(salary, '$99,999.00') SALARY FROM employees WHERE last\_name = 'Ernst';

Convert a character string to a number format using the TO\_NUMBER function: TO\_NUMBER(char[,'format\_model'])

SELECT last\_name, TO\_CHAR(hire\_date, 'DD-Mon-YYYY') FROM employees WHERE hire\_date < TO\_DATE('01-Jan-90','DD-Mon-RR');

**Nesting Functions**:

SELECT last\_name,UPPER(CONCAT(SUBSTR (LAST\_NAME, 1, 8), '\_US')) FROM employees WHERE department\_id = 60;

**General functions:**

**NVL Functions:** whenever there is a null value , we can replace the null values with given values using NVL functions.

Converts a null value to an actual value: Data types that can be used are date, character, and number.

Data types must match:

NVL(commission\_pct,0)

• NVL(hire\_date,'01-JAN-97')

• NVL(job\_id,'No Job Yet')

**Ex**. select commission\_pct,**nvl**(commission\_pct, '2') new from hr.employees;--- its returns ‘2’ where the value is null other not null values remains same.

SELECT last\_name, salary, NVL(commission\_pct, 0),(salary\*12) + (salary\*12\*NVL(commission\_pct, 0)) AN\_SAL FROM employees;

**nvL2**: takes two values in function..one for the null value and other for the not null value, both can be replaced. first value updates the not null value and second value changes the null values.

**Ex**. select commission\_pct,**nvl2**(commission\_pct,0.5,3) changed from hr.employees; --- return the value ‘3’ where ever we have null values and ‘0.5’ is returns where the value is not null.

**Ex**. SELECT last\_name, salary, commission\_pct, NVL2(commission\_pct,'SAL+COMM', 'SAL') income FROM employees WHERE department\_id IN (50, 80);

**Nullif**:when both values are equal then give the result as “null”. is used to compare two values based on the comparision, if the condition is true then value is changed to null. **Null if they are equal.**

**Ex.** SELECT first\_name, LENGTH(first\_name) "expr1",last\_name, LENGTH(last\_name) "expr2", NULLIF(LENGTH(first\_name), LENGTH(last\_name)) result FROM employees;

**COALESCE:**The advantage of the COALESCE function over the NVL function is that theCOALESCE function can take multiple alternate values.

– If the first expression is not null, the COALESCE function returns that expression;otherwise, it does a COALESCE of the remaining expressions.

**Ex.** SELECT last\_name,COALESCE(manager\_id,commission\_pct, -1) comm FROM employees ORDER BY commission\_pct;--- check the first expressions is not null then returns that value, if fexpr null and second is not null then returns second value, if both are null then returns the given value.

**Conditional Expressions:**

**Syntax:** CASE expr WHEN comparison\_expr1 THEN return\_expr1

[WHEN comparison\_expr2 THEN return\_expr2

WHEN comparison\_exprn THEN return\_exprn

ELSE else\_expr]

Ex. SELECT last\_name, job\_id, salary,

CASE job\_id WHEN 'IT\_PROG' THEN 1.10\*salary

WHEN 'ST\_CLERK' THEN 1.15\*salary

WHEN 'SA\_REP' THEN 1.20\*salary

ELSE salary END "REVISED\_SALARY"

FROM employees;

2.**Syntax:** DECODE(col|expression, search1, result1[, search2, result2,...,][, default])

Ex. SELECT last\_name, job\_id, salary,DECODE(job\_id, 'IT\_PROG', 1.10\*salary,'ST\_CLERK', 1.15\*salary,'SA\_REP', 1.20\*salary, salary) REVISED\_SALARY FROM employees;

Decode is same like Case without any when then and else keywords, it just takes comma seperated values.

**GROUP** **Functions**: Group functions operate on sets of rows to give one result per group. The column in select, should be part of either group by or aggregate functions. Group functions are multi row functions, meaning they perform on multiple rows and gives the single row answer.

**AVG**:

SELECT AVG(commission\_pct) FROM employees;

SELECT AVG(NVL(commission\_pct, 0)) FROM employees;

**COUNT**: COUNT(\*) returns the number of rows in a table:

Ex. SELECT COUNT(\*) FROM employees WHERE department\_id = 50;

SELECT COUNT(commission\_pct) FROM employees WHERE department\_id = 80;

**COUNT**(**DISTINCT** expr) returns the number of distinct non-null values of the expr.

Ex. SELECT COUNT(DISTINCT department\_id) FROM employees;

– **MAX and MIN**: can use MIN and MAX for numeric, character, and date data types.

Ex. SELECT MIN(hire\_date), MAX(hire\_date) FROM employees;

**STDDEV:** standard deviation

SUM:

Ex. SELECT AVG(salary), MAX(salary),MIN(salary), SUM(salary) FROM employees WHERE job\_id LIKE '%REP%';

VARIANCE:

Group functions ignore null values in the column:

SELECT AVG(commission\_pct) FROM employees;

SELECT AVG(NVL(commission\_pct, 0)) FROM employees;

Display the maximum average salary: we can do nesting of group functions:

Ex. SELECT MAX(AVG(salary)) FROM employees GROUP BY department\_id;

**Group By:** Syntax.

SELECT column, group\_function(column) FROM table [WHERE condition][GROUP BY group\_by\_expression][ORDER BY column];

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

Ex. SELECT department\_id, AVG(salary) FROM employees GROUP BY department\_id ;

The GROUP BY column does not have to be in the SELECT list.

Ex. SELECT AVG(salary) FROM employees GROUP BY department\_id ;

Group By Cluase can be used on multiple columns.

Ex. SELECT department\_id dept\_id, job\_id, SUM(salary) FROM employees GROUP BY department\_id, job\_id ;

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause:

SELECT department\_id,sum(salary) FROM hr.employees ;--- gives error : “not a single grp function”

We cannot use the WHERE clause to restrict groups. Use the HAVING clause to restrict groups. Group functions cannot be used in the WHERE clause.

SELECT department\_id, AVG(salary) FROM employees **WHERE AVG(salary**) > 8000 GROUP BY department\_id;--- ERROR

When we use the **HAVING** clause, the Oracle server restricts groups as follows:

1. Rows are grouped.

2. The group function is applied.

3. Groups matching the HAVING clause are displayed.

Syntax: SELECT column, group\_function FROM table [WHERE condition] [GROUP BY group\_by\_expression] [HAVING group\_condition]

Ex. SELECT department\_id, MAX(salary)

FROM employees GROUP BY department\_id

HAVING MAX(salary)>10000 ;

Ex. SELECT job\_id, SUM(salary) PAYROLL

FROM employees WHERE job\_id NOT LIKE '%REP%'

GROUP BY job\_id HAVING SUM(salary) > 13000 order by SUM(salary);

**JOINS:** Why we need joins. JOIN is used to combine rows from two or more tables based on a common field between them. Types:

1. natural join
2. inner join
3. outer join
   1. Left outer join
   2. Right Outer join
   3. Full Outer Join
   4. Cross join

Natural Join: gives matching records based on the same common columns, same values and data types. “Where” condition can’t be given in natural joins.

The NATURAL JOIN clause is based on all columns in the two tables that have the same name.

It selects rows from the two tables that have equal values in all matched columns.

If the columns having the same names have different data types, an error is returned.

Ex. SELECT department\_id, department\_name, location\_id, city FROM departments NATURAL JOIN locations ;

Select \* from hr.departments NATURAL JOIN hr.locations ;

**INNER JOIN/ JOIN: inner join can be done ony using either “using” clause or with “on” keyword… and the using caluse column name must be specified in parathesis.**

If we want to specify particular condition, then we use clause “using” with inner join. Using clause condition column must be common in both tables.

select \* from hr.departments join hr.employees using (manager\_id);

select \* from hr.departments inner join hr.employees using (department\_id); --- it searches based on particular column (department\_id) in both tables and returns the data.---without using with natural join it searches for all the common columns in both tables.

Ex. Select hr.employees.first\_name, hr.employees.salary, hr.departments.department\_name, hr.departments.department\_id from hr.departments join hr.employees using(manager\_id);

When using clause “using” we cant use alias names , and column name and data types should always match. To overcome this problem we use “on” instead of “using” to use alias names, because of alias names it improves the performance.

**Inner join with “on”:**

select e.first\_name,e.salary,d.department\_name,d.department\_id from hr.departments d join hr.employees e on (d.manager\_id=e.manager\_id);

with on” we can use alias names and we can use any column names for conditions, alias names increases the performance.

Self Join: if we want to establish a relation between the same tables then we call it as self join…join between two tables alias names would be different for easy understand.

Ex. SELECT e.last\_name emp, m.last\_name mgr FROM employees e **JOIN** employees m ON (e.manager\_id = m.employee\_id);--equijoin

Ex. SELECT e.employee\_id, e.last\_name, e.department\_id, d.department\_id, d.location\_id FROM employees e JOIN departments d ON (e.department\_id = d.department\_id) AND e.manager\_id = 149 ; ---(additional conditions)

**Three-Way join on “ON” clause: join with more than two tables : three diff columns from three diff tables**

Ex. SELECT employee\_id, city, department\_name FROM employees e **JOIN** departments d ON d.department\_id = e.department\_id **JOIN** locations l ON d.location\_id = l.location\_id;

Non-Equi Join:

Ex. SELECT e.last\_name, e.salary, j.job\_title FROM hr.employees e JOIN hr.jobs j ON e.salary BETWEEN j.min\_salary AND j.max\_salary;

An **equi-join** is a specific type of comparator-based join, that uses only [equality](https://en.wikipedia.org/wiki/Equality_(mathematics)) comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

**Outer Join**:

**Full Outer Join**: It gives all the matching columns from both tables and also gives unmatched columns from both tables.(it gives all the common rows from left and right and also unmatched rows from both tables).

select count(\*) from hr.departments full outer join hr.employees using(manager\_id);

select \* from hr.departments full outer join hr.employees using(department\_id);

select count(\*) from hr.departments full outer join hr.employees using(department\_id);

**Left Outer Join:** it gives all the common rows between left and right, and also gives unmatched rows from left side table before the “join” keyword.

**Ex.** SELECT e.last\_name, e.department\_id, d.department\_name FROM hr.employees e **LEFT OUTER JOIN** hr.departments d ON (e.department\_id = d.department\_id) ;-- it displays common rows from both tables and also the unmatched rows from the employees table.

Right Outer Join: it gives all the common rows from both tables and also gives unmatched rows from the right side table mentioned after the “join” keyword.

Ex. SELECT e.last\_name, e.department\_id, d.department\_name FROM employees e **RIGHT OUTER JOIN** departments d ON (e.department\_id = d.department\_id) ; --- displays the all the common rows from both tables along with unmatched rows from departments table.

select count(\*) from hr.locations l full outer join hr.departments d on (l.location\_id = d.location\_id);

select count(\*) from hr.locations l left outer join hr.departments d on (l.location\_id = d.location\_id);

select count(\*) from hr.locations l right outer join hr.departments d on (l.location\_id = d.location\_id);

The CROSS JOIN clause produces the cross-product of two tables.This is also called a Cartesian product between the two tables.

A Cartesian product is formed when:

• A join condition is omitted

• A join condition is invalid

• All rows in the first table are joined to all rows in the second table

Ex. SELECT last\_name, department\_name FROM hr.employees CROSS JOIN hr.departments;

Sub Query is inside a query or inner query. Sub query is evaluated first and the result is used in the main query.

Single row operators (=,>,<,>=,<=,<>) are used with single row sub queries.

Multi row operators (in, any and all) should be used with multi row sub queries.

In: it should satisfy the given conditions

Any: it checks for any value among the result like OR

All:it should satisfy all the conditions like AND

Null values in sub queries returns no rows in main queries.

SET Operators: are performed in select queries…used to combine the results of queries. The colums selected in both queries hasto be same columns and of same data types, they can be rom different tables also. We can use ordery by but it should be used at the end of the statements.

1. Union
2. Union All
3. Minus
4. Intersection

Union: union combines to select statements outputs without duplicates. The UNION operator returns results from both queries after eliminating duplications.

Ex. SELECT employee\_id, job\_id FROM employees UNION SELECT employee\_id, job\_id FROM job\_history;

Union all: combines two select statements outputs without omitting the duplicates … all rows are displayed. The UNION ALL operator returns results from both queries, including all duplications.

Ex. SELECT employee\_id, job\_id, department\_id FROM employees UNION ALL SELECT employee\_id, job\_id, department\_id FROM job\_history ORDER BY employee\_id;

Intersection: displays the common rows from both statements. The INTERSECT operator returns rows that are common to both queries.

Ex. SELECT employee\_id, job\_id FROM employees INTERSECT SELECT employee\_id, job\_id FROM job\_history;

Minus: it just gives the rows after subtracting from one another . The MINUS operator returns rows in the first query that are not present in the second query.

Ex. SELECT employee\_id FROM employees MINUS SELECT employee\_id FROM job\_history;

Set operator guidelines:

The expressions in the SELECT lists must match in number and data type.

– Parentheses can be used to alter the sequence of execution.

– The ORDER BY clause:

• Can appear only at the very end of the statement

• Will accept the column name, aliases from the first SELECT statement, orthe positional notation

- Duplicate rows are automatically eliminated except in UNION ALL.

– Column names from the first query appear in the result.

– The output is sorted in ascending order by default except in UNION ALL. Use UNION to return all distinct rows

– Use UNION ALL to return all rows, including duplicates

– Use INTERSECT to return all rows that are shared by both queries

– Use MINUS to return all distinct rows that are selected by the first query but not by the second

– Use ORDER BY only at the very end of the statement.

**Constraints**: are the rules /restrictions which are given for the columns.

**Primary** key: always unique and not null

**Foreign** key: referential intergrity: FK in one table in one table is the primary key in other table, fk is used to establish the relation between two tables using common column.

Ex. Department\_id is the primary key in departments table and is the FK for the employees table.

**Not null**: it do not accept null values

**Unique**: it should always be distinct –duplicate values are not allowed.

DDL:create,alter,

Create: is used to create the table.

Alter: is used to alter the table..like we can add the columns,modify the columns and delete the columns . while adding the columns we need to give the datatypes and constarints(if any).

Drop

Truncate:

DML: select, insert, update, delete

Insert: is used to insert the records into the table…we can insert all the values at a time or we can choose particular columns and insert the values.

DCL: revoke and grant

TCL: savepoint,rollback and commit

Database Objects:

Tables: set of rows and columns

View: is something which is like virual tables

Sequence: it generate sequence of numbers which can be used to insert data into any numbered columns.

**Questions and doubts:**

1. Explain about to\_char and to\_date : in (to\_char) we specify what format we want for conversions and in(to\_date) in what format we representing the date in the parantheses.converting varchar to date format.it justconverts the data type.
2. Diff between char,varchar and varchar2
3. Diff between nvl,nvl2,nullif,coalesce---practice
4. Group functions –group by clause
5. Conditional practice

Assignment queries:

1. Write a query to select first\_name,department name,deptid,salary from employees and departments table based on all common columns.

ANS. select first\_name,department\_name,department\_id,salary from hr.employees natural join hr.departments;

1. Write a query to select first\_name,department name,deptid,salary from employees and departments table based on departmentid.

Ans. select first\_name,department\_name,department\_id,salary from hr.employees join hr.departments using (department\_id);

1. Write a query to select City, firstname, department name for an employee
2. Write a query to Fetch empname, departmentname and deptid with and without employees.

select first\_name,department\_name,department\_id from hr.employees right outer join hr.departments using (department\_id);

1. Find the employee with second highest salary
2. Fetch only the 10 records from the employee table
3. Write a query to fetch all the employees who have the same salary as the minimum salary of sales department.-- select \* from hr.employees where salary = (select min(salary) from hr.employees group by department\_id having (department\_id=80));
4. All departments based out of location us---
5. Which employees have salaries greater than Abel’s salary?

10.Write a query to find employees whose salary is same as employee 141 and whose job id is same as 143.

11. W query to find a department whose min salary is greater than min salary of sales department?--- select min(salary),department\_id from hr.employees group by department\_id having min(salary) > (select min(salary) from hr.employees where department\_id=80);

12. write a query to select first name and hire date of employees who were hired after employee smith

13. to find employees who report to king?

14. query to find Top salaried employee details in each department

15. query to find Employee details of second highest salaried person

16. query to find min salary under each job category in specific department

|  |  |
| --- | --- |
|  | display location\_id, city and country id by ordering cities in asc order and countries in desc order |
|  | employees who is an IT programmer and who has more than 5 years of experience |
|  | employees whose salary>5000 and who are sales rep's or employees who belong to dept-60 |
|  | Write a query to Select first\_name, dept name, deptid,salary from employees table and departments table based on all the common column. |
|  | Write a query to Select first\_name, dept name, deptid,salary from employees table and departments table based on dept id |
|  | Write a query to Select city, firstname, deptname for an employee |
|  | Write a query to fetch empname, deptname and deptid of all the departments with and without employees. |
|  | query to find min sal under each job category in specific department. |
|  | write a query to find location\_ids common to department and locationt able |
|  | write a query to fetch employee firstname and street adress (use emp table and locations table) |

7th highest salry in employees table

write a query to get employee firstname, department id, salary of all employees who have same salary as the min salary of department with id 50

get all employees who have same job\_id as the employee with id 141 and salary is same as employeee with id 143

employers who are not managers in employees table

employers who are highest paid in the company

employers who are least paid in the company

create a table and modify column name , datatype of column

Create a table and execute insert,delete,drop,flashback,insert,truncate and see the difference

-------------------------------------Sql Mocks----------------------------------

## SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

* [**NOT NULL**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

#### Explain the difference between rowid and rownum.

ROWID is a unique pseudo number assigned to a row. Rownum returns the number of rows for a resultant query. BOTH are pseudo columns not occupying any physical space in the database.

Example of Rownum:  
Sort salary and return first 5 rows.  
select \* from  
( select \*  
       from emp   
       order by sal desc )   
where ROWNUM <= 5;

Example of Rowid:  
Below query selects address of all rows that contain data for students in department 20  
SELECT ROWID, last\_name  
       FROM student  
       WHERE department\_id = 20;

RowId represents a row in a table internally. It can be used for fast access to the row. Rownum is a function of the result set.   
select \* from Student where rownum = 2 will get the first 2 rows of your result set.

Duff between delete and truncate?