SQL

more info:https://github.com/medhawbl/MarchClassDocs/blob/master/sql2-classqueries.txt

SQL

Data -

Departments, products, suppliers, customers, orders,…

Loans, Account, transactions, customer

Employees,department

Fastrack, 437854398758, ang@g.com....

Information –

Stuctured and organized data.

DataBase- System that stores information/organized data.

Files- Excel, xml,word

MsAccess

Oracle

Oracle MySql

Microsoft Sql Server

IBm Db2

Postgress

NoSQl DB:

MongoDB

Cassandra

DBMS – DataBaseManagement System is an application/software which helps

To analyse and capture data – It can interact with User, Application and DataBase itself

To perform CRUD(Create, retrieve, update, delete) Operations.

Amazon- Departments, products, suppliers, customers, orders

Create tables - columns, constraints

Insert data into the table

Update the information

Delete the information.

Normalization- is the technique that helps to avoid data redundancy and maintain data integrity.

RDBMS : RelationalDatabase Management System uses normalization principles.

RDBMS maintains relation between tables using primarykey, foreignkey, index…

Table – It is one of the Database Objects. Set of rows and columns

Column- attribute

SQL – StructuredQueryLanguage that interacts with DataBase to perform CRUD operations.

ER Diagrams – pictorial representation of relations between tables in RDBMS.

Types of Relations:

One-One:one customer having one mobile number ,or every employee belongs to one department

One-Many:one customer may be having more than one order.

Many-Many: an order can have multiple products and a product can be in multiple orders

primaryKey – A column or Set of Columns which help to identify a row uniquely.PK cannot be null or empty

In a table. Max of 32 columns we can combine I oracle DB as PK.

ForeignKey – A column that enforces referential integrity. A column in one table(child table)

that references column in another table(parent table).In simpler words, the **foreign key**is defined in a second table, but it refers to the primary **key** or a unique **key** in the first table.

Create table tablename (

Name varchar2 primary key,

Id number primary key

)

<https://login.oracle.com/mysso/signon.jsp-> Create account

USe the same uname and pwd for <https://livesql.oracle.com/apex/livesql/file/index.html>

If u are not creating account use:

username: training@whitebox-learning.com ,

password: Innovapath123

Schema – It is a structure/space where all database objects are created.

Toad,SQLDeveloper,RoboMongo – tools which help to connect to

Given database and execute the queries/methods.

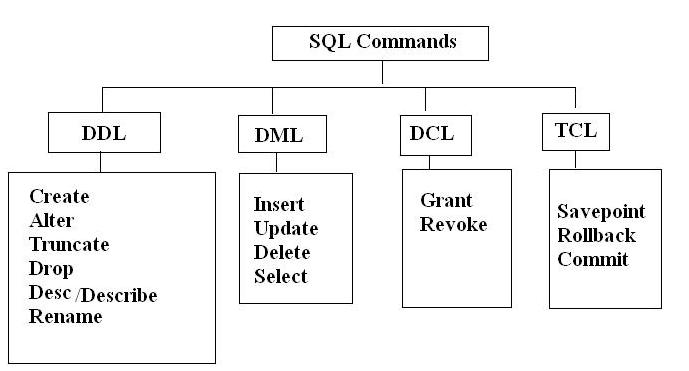
But we are using livesql session to execute queries.

DDL

DML - Select

DCL

TCl



**DML**

DML is abbreviation of **Data Manipulation Language**. It is used to retrieve, store, modify, delete, insert and update data in database.

Examples: SELECT, UPDATE, INSERT statements

**DDL**

DDL is abbreviation of **Data Definition Language**. It is used to create and modify the structure of database objects in database.

Examples: CREATE, ALTER, DROP statements

**DCL**

DCL is abbreviation of **Data Control Language**. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

Examples: GRANT, REVOKE statements

**TCL**

TCL is abbreviation of **Transactional Control Language**. It is used to manage different transactions occurring within a database.

Examples: COMMIT, ROLLBACK statements

SQL is not case sensitive but data is case sensitive

DataType: Kind of data we access or store

Number

Varchar – string

Date

BLOB

Select:

Syntax: Select columnnames from tablename;

\*- all columns

More than one column- , separated names

EX:

***select first\_name,last\_name,salary from hr.employees***

Where- to filter the rows

**EX:*select first\_name,last\_name,salary from hr.employees where first\_name='Steven'***

Order by- sort in asc or desc order

Can be used for more than one column, it will always be at the end of sql statement

***EX:select first\_name,last\_name,salary from hr.employees order by salary desc-→for decending***

***select first\_name,last\_name,salary from hr.employees order by salary asc--→ascending***

we can also sort by two colums (it has to be logical)like we can sort usig employee id and department id -→which will first sort by employee id and then sort by department id

***EX:***

***select \*from hr.employees where salary >10000 order by salary,department\_id;***

Distinct- get unique column data

***EX:select distinct last\_name from hr.employees;***

Like with %(your can use it in starting or ending )

***EX:***

***select \* from hr.employees where last\_name like 'K%';*** -→which startswith ‘K’

***select \* from hr.employees where last\_name like '%K';*** -→which endswith ‘K’

***select \* from hr.employees where last\_name like '%K%';*** -→which in between ‘K’

**select \* from hr.employees where first\_name like '\_g%' --→underscore for one char**

And – for more than one condition

***EX:select \* from hr.employees where first\_name like 'A%' and last\_name like 'K%';***

OR:

***EX:select \* from hr.employees where first\_name like 'A%' OR last\_name like 'K%';***

between :

***EX:select \* from hr.employees where employee\_id between 100 and 150;***

arithmetic - \*,/,+,-

EX:***select first\_name,last\_name,(salary+100) as temp from hr.employees;***

checkfor null :EX

***select first\_name ,salary from hr.employees where COMMISSION\_PCT is not null;***

***select first\_name ,salary from hr.employees where COMMISSION\_PCT is null;***

concatination : uses ‘||’ to concatinate

***select first\_name||' '||last\_name as Full\_Name, salary from hr.employees;***

***select first\_name||' '||last\_name as Full\_Name,q'[its department is:]'||department\_id from hr.employees;--→for adding a sentence***

to describe table: EX:

***describe employees***

conditional-

= single value comparision

In- (multiple equal to condition )multiple value comparision

EX:

***select \* from hr.employees where job\_id in ('AD\_PRES','IT\_PROG');***

<> single value not comparision:

select \* from hr.employees where job\_id <>'AD\_PRES’

Not in(multiple not equal to condition ): EX: for more values

***select \* from hr.employees where job\_id not in ('AD\_PRES','IT\_PROG');***

<

<=

>

>=

upper case:

***EX:select UPPER (first\_name||' '||last\_name) as Full\_Name from hr.employees***

*lower case:*

***EX:select lower (first\_name||' '||last\_name) as Full\_Name from hr.employees***

initials capital:

***EX:select initcap (first\_name||' '||last\_name) as Full\_Name from hr.employees***

CONCAT:

EX:***select concat (first\_name,last\_name) as Full\_Name from hr.employees;***

*length:*

EX:**select first\_name,length(first\_name)from hr.employees;**

SUBSTR:

EX:***select first\_name, substr(first\_name,2,3) from hr.employees;***

***where 2=index***

***and 3=count***

***-->output***

|  |  |
| --- | --- |
| **FIRST\_NAME** | **SUBSTR(FIRST\_NAME,2,3)** |
| Ellen | lle |
| Sundar | und |

instr:

EX:

***select first\_name ,instr(first\_name,'a') from hr.employees;***

|  |  |
| --- | --- |
| **FIRST\_NAME** | **INSTR(FIRST\_NAME,'A')** |
| Ellen | 0 |
| Sundar | 5 |
| Mozhe | 0 |

Replace :

***EX:select job\_id ,replace(job\_id,'AD','AJ')from hr.employees;***

---->output:

|  |  |
| --- | --- |
| **JOB\_ID** | **REPLACE(JOB\_ID,'AD','AJ')** |
| AC\_ACCOUNT | AC\_ACCOUNT |
| AC\_MGR | AC\_MGR |
| AD\_ASST | AJ\_ASST |
| AD\_PRES | AJ\_PRES |

Trim:removes from the ends

***select first\_name,trim('E'from first\_name) from hr.employees;***

***--→output***

|  |  |
| --- | --- |
| **FIRST\_NAME** | **TRIM('E'FROMFIRST\_NAME)** |
| Ellen | llen |

round :

***EX:select round(34.567,2)from dual;***

|  |
| --- |
| **ROUND(34.567,2)** |
| 34.57 |

***select round(20037.9937)from dual;-->20038***

***select round(20037.9937,-1)from dual;-->20040***

|  |
| --- |
| **ROUND(20037.9937,-1)** |
| 20040 |

trunc:

***EX:***

***select trunc(34.5633447,3),trunc(20037.9937),trunc(20037.9937,-1)from dual;***

***output:***

|  |  |  |
| --- | --- | --- |
| **TRUNC(34.5633447,3)** | **TRUNC(20037.9937)** | **TRUNC(20037.9937,-1)** |
| 34.563 | 20037 | 20030 |

mod:

***EX:***

***select mod(200,100)from dual;-->0***

DATE:

months\_between:

***EX:SELECT MONTHS\_BETWEEN***

***(TO\_DATE('01-01-1998','MM-DD-YYYY'),***

***TO\_DATE('02-02-1995','MM-DD-YYYY') ) "Months"***

***FROM DUAL;***

***NEXT DAY:***

SELECT NEXT\_DAY('02-FEB-2001','TUESDAY') "NEXT DAY"

FROM DUAL;

ROUND Date:

EX: SELECT ROUND (TO\_DATE ('27-OCT-00'),'YEAR')

"New Year" FROM DUAL;

New Year

---------

01-JAN-01

***trunc date:***

Ex:

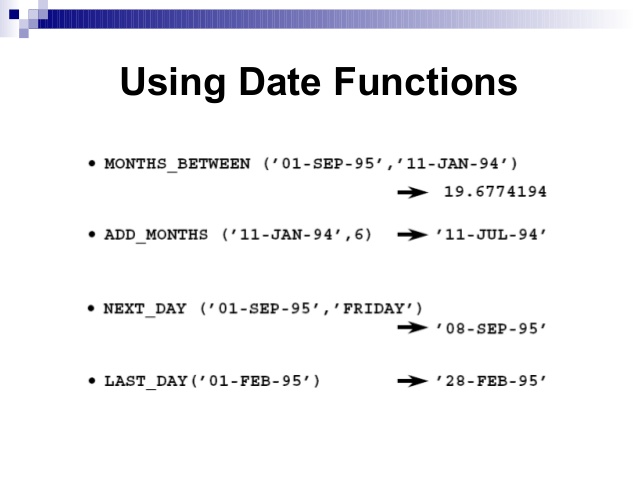
SELECT TRUNC(TO\_DATE('27-OCT-92','DD-MON-YY'), 'YEAR')

"New Year" FROM DUAL;

New Year

---------

01-JAN-92



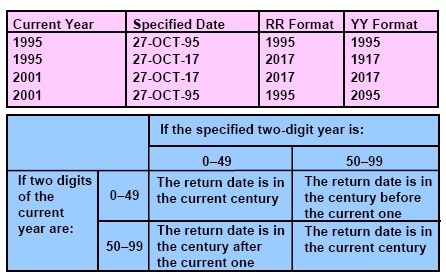
TO\_CHAr

ex:

***-select first\_name,to\_char(HIRE\_DATE,'dd/mm/yy') from hr.employees;***

***-select to\_char(salary )from hr.employees;***

***-select to\_char(salary,'$99,999.00' )from hr.employees;--→’$99,999.00'’-→being the format***

******

to\_date:

***EX:SELECT***

***(TO\_DATE('01-01-1998'(\*the date in char form),(format)'MM-DD-YYYY')"Months"***

***FROM DUAL;***

SingleRowFunctions : Operate on every row and result is also for each row

MultiRowFunctions: Operate on multiple rows and gives a single result

DateFormat:

dd-mon-rr - this is default date format

12-02-17 – 2017?1917?

todays date is in between 0-49 of current century – means we are in first half of century

and you are trying to insert 12-01-95 to hire\_date then it will take prev

century.

so o/p here is - 12-01-1995

to\_char( datedatatype, ‘format which we want see in o/p’);

to\_char(sysdate, ‘dd-mm-yyyy’)

to\_date(chardatatype, ‘ format what we are giving in input’)

o/p will be in default date format

to\_date(’12-06-2017’, ‘dd-mm-yyyy’)

***null functions:***

nvl(columnname, substitutionForNullvalue) - if column value is null then take the substitutionForNullvalue as output.

***EX:select first\_name,nvl(COMMISSION\_PCT,0) from hr.employees;-→so if COMMISSION\_PCT is null then replace it with 0***

***anothet example:***

***select first\_name,(nvl(COMMISSION\_PCT,0)\*23) from hr.employees;***

nvl2(colName, value1, value2) – if col value is null take value 2 in o/p else take value 1 in o/p.

nullif(col1,col2) – if both values of col1 and col2 are equal return null in o/p else return col1 value

***EX:***

***select first\_name,nullif(length(first\_name),length(last\_name)) from hr.employees;***

output:

|  |  |
| --- | --- |
| **FIRST\_NAME** | **NULLIF(LENGTH(FIRST\_NAME),LENGTH(LAST\_NAME))** |
| Ellen | 5 |
| Sundar | 6 |

coalesce(col1, col2…,..substitutionForNullvalue) – It will take first not null value as o/p from the give set of values or go with the subsitute value

***EX:select coalesce(manager\_id,department\_id-1) from hr.employees;***

***NOTE:do case -when then and decode***

**Group** Functions – Operate on multiple rows and give a single result

eg: avg,min,max,sum,count

EX:

***select avg(salary),min(salary),max(salary),sum(salary),count(\*) from hr.employees;***

***output-->***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AVG(SALARY)** | **MIN(SALARY)** | **MAX(SALARY)** | **SUM(SALARY)** | **COUNT(SALARY)** |
| 6461.831775700934579439252336448598130841 | 2100 | 24000 | 691416 | 107 |

**You can use a group function with another group function → like the abouve example because the output will be one but cannot use single function with group function since single row may have more than one output.**

**groupby** having – group by is used to group the set of rows and then perform and group function operation on them

Ex:

***select max(salary) from hr.employees group by job\_id***

***--select distinct job\_id,max(salary) from hr.employees group by job\_id ;***

***output-->***

|  |  |
| --- | --- |
| **JOB\_ID** | **MAX(SALARY)** |
| IT\_PROG | 9000 |
| AC\_MGR | 12008 |
| AC\_ACCOUNT | 8300 |

--***select count(first\_name),manager\_id from hr.employees group by manager\_id;-→print the number of employee under a particular manager.***

***-→output:***

|  |  |
| --- | --- |
| **NUMBER\_OF\_EMPLOYEES** | **MANAGER\_ID** |
| 1 | - |
| 14 | 100 |
| 8 | 123 |
| 8 | 120 |

***--select avg(salary) ,department\_id from hr.employees group by department\_id;***

-->output:

|  |  |
| --- | --- |
| **AVG(SALARY)** | **DEPARTMENT\_ID** |
| 8601.333333333333333333333333333333333333 | 100 |
| 4150 | 30 |
| 7000 | - |
| 19333.3333333333333333333333333333333333 | 90 |

***--select max(salary),department\_id from hr.employees group by department\_id***

***-→output:***

|  |  |
| --- | --- |
| **MAX(SALARY)** | **DEPARTMENT\_ID** |
| 12008 | 100 |
| 11000 | 30 |

***--select max(salary),department\_id,job\_id from hr.employees group by department\_id,job\_id order by department\_id;***

***-→output:***

|  |  |  |
| --- | --- | --- |
| **MAX(SALARY)** | **DEPARTMENT\_ID** | **JOB\_ID** |
| 4400 | 10 | AD\_ASST |
| 13000 | 20 | MK\_MAN |
| 6000 | 20 | MK\_REP |

eg: find no of products by Hp/dell/… individually in amazon

min cost of product under each supplier individually in amazon

costliest mobile under each manufacturer like samsung, apple , google

When we are using group by the condition is:

Always columns in select statement should be part of either a group function or part of group by.

eg:

wrong query: bcz employee\_id is a separate column which is not part of group by or group function

select avg(salary),department\_id,employee\_id from hr.employees group by department\_id;

correct query:

select avg(salary),department\_id from hr.employees group by department\_id;

group by can be used for more than one column, in this case

it will first group by firstcolumn and within that group it again

groups by second column and so on….

**having** : will be used along with group by to limit the rows based on condition as we cannot use where after group by.

eg: this gives an error

***select department\_id, sum(salary),min(salary) from hr.employees group by department\_id where min(salary) >5000;***

Also we cannot use group functions with where clause:

eg: this gives an error

***select \* from hr.employees where min(salary) >3000***

These are valid queries:

***select department\_id, sum(salary),min(salary) from hr.employees group by department\_id having min(salary) >5000;***

***select department\_id, sum(salary),min(salary) from hr.employees where department\_id in (90,100)***

***group by department\_id having min(salary) >5000;***

Joins : Combining 2 or more tables to get data required from multiple

tables.

**InnerJoin** :

**NaturalJoin**- Joins more than one table based on all common columns between the tables provided columnname and datatype of column is same.

**Natural Join with using**- Joins more than one table based specific common columns between the tables provided columnname and datatype of column is same.

**Note::when you are using join… using we cannot use natural join**

they both are mutually exclusive.

**Join with on** – this takes a common column condition even if columnname is different.

Ex:

***eg: select empname,deptid,deptname from hr.emp join hr.depts***

***on emp.dept\_id= depts.department\_id;***

Example of multiple inner join:

***--select e.first\_name,e.last\_name ,l.city ,c.country\_name from hr.employees e***

***inner join hr.departments d on e.department\_id=d.department\_id***

***join hr.locations l on d.location\_id= l.location\_id***

***join hr.countries c on l.country\_id=c.country\_id;***

**OuterJoin:**

**FullOuterJoin-** Matched rows between the tables we are joining and also the unmatched rows.

**LeftOuterJoin:** Matched rows between the tables and unmatched rows from left table.

**RightOuterJoin:** Matched rows between the tables and unmatched rows from right table.

SelfJoin – Joining table with itself using common column.

Ex:

***select (e.first\_name||' '||e.last\_name)as EmployeeName ,(m.first\_name||' '||m.last\_name)as ManagerName from hr.employees e***

***join hr.employees m***

***on(e.manager\_id=m.employee\_id);***

**CrossJoin**- Cartesian product

hr.emp-20

hr.dept-30

***select \* from hr.emp, hr.dept***

***select \* from hr.emp cross join hr.dept***

***result- 20\*30= 600***

EquiJoin- join based on equality of common columns

e.dept\_id=d.dept\_id

NonEquiJoin: Join based on non equality condition of common columns

eg: e.sal between g.min\_sal and g.max\_sal

**SetOperators** – To join 2 select statements provided both select statements have same column names and same datatype

**Union** - Gives all the rows from 2 query results without duplicates

**UnionAll** - Gives all the rows from 2 query results along with duplicates

**Intersect** - Gives common rows from 2 query results

**Minus** – subtracts one query result from other and gives remaining rows.

diff b/n Union and Union All

<https://github.com/medhawbl/MarchClassDocs>

[training@whitebox-learning.com](mailto:training@whitebox-learning.com)

SubQueries:

***EX:select first\_name,last\_name,salary from hr.employees where salary = (select min(salary) from (select salary from (select distinct salary from hr.employees order by salary desc***

***)where rowNum<=6))***

***-→selects the name of person with 6th highest salary.***

***--select employee\_id ,first\_name,last\_name from hr.employees where employee\_id in (select distinct manager\_id from hr.employees where manager\_id is not null)***

***-→to get the managers name and details***

Query inside the query, result of sub query will be the input to main query

And always subquery executes first.

single row operators(<,<=,=,<>,!=,>,>=) can be used when subquery returns one row as the result.

Multi row operators (in, any,all) can be used when subquery

returns more than one row.

in

any – like or condition

all – like and condition

rownum – which gives a unique number for every row in select query,

and rownum will be assigned after predicate(where ) of query is executed and before order by clause .

you cannot give rownum= somevalue or rownum> somevalue

we can give rownum<=somevalue

varchar2 – cannot differentiate between null and blank.

Constraints: setting rules on data being inserted into that column.

PrimaryKey – unique + not null

ForeignKey

Unique

Not null

check…

DDL –

create

Alter

Drop

Truncate

DML-

Insert

update

Select

delete

To insert id column using a sequence instead of hardcoding the id everytime:

Create table

Create sequence

create trigger on the insert operation for table

Insert data for other columns- id will automatically inserted by trigger.

eg:

create table student(

sid number(20) primary key,

name varchar2(20) );

create or replace trigger seqTrigger2

before insert on student

for each row

begin

:new.sid:=idseq.nextval;

end;

insert into student(name) values('joe');

select \* from student;

more ex:https://github.com/medhawbl/MarchClassDocs/blob/master/sql3-classqueries.txt

Both drop and truncate are ddl operations:

Drop – delete all the data along with table structure

They do not fire triggers like delete.

Truncate – delete all the data but not table structure,

truncate is faster than delete bcz it does not cache the data.

They do not fire triggers like delete.

Delete is a dml operation:

Delete will delete the data not the table structure, we can

have a where condition to restrict the rows to be deleted.

delete will fire the triggers before deleting the data.

commit--

rollback

Latest oracle versions- there is option to get back table after drop:

By using flashback:

flashback table student to before drop