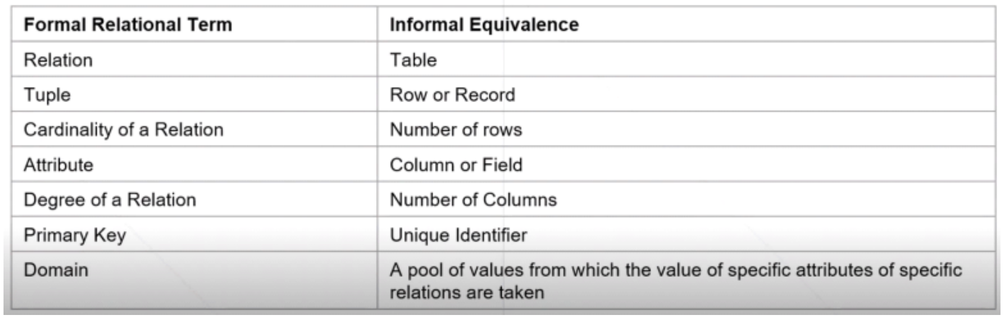
DBMS Class Notes

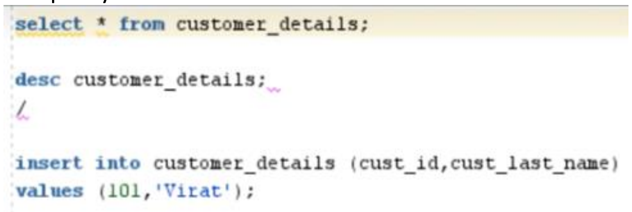
Monday, March 14, 2022 11:49 AM

- Database : we store the data in a structured or unstructured manner, database is a organized collection of interrelated data.
- DBMS: Software which helps us to maintain and manipulate the database.
- DBMS is a collection of interrelated files and a set of programs that allow users to access and modify these files. The main goal of DBMS is to provide a convenient and efficient way to store, retrieve and modify information. These are defined to defined structures to store the data and provide mechanism to manipulate data. Helps us in case of system crashes or attempts of unauthorized access. And share data among the different users.
- Why we need DBMS: DBMS establishes concurrency: for example, two or more users can access or modify the database at a time. We can establish business constraints for the attributes that we need. Controls redundancy and inconsistency. Provides secure access to the database. Enable back up recovery options.
- We organize the data of the database using data models. Data model is a conceptual tool. Which establish relationship among the data either semantically or using constraints. Object data model , relational data model , hierarchical data model are some examples of data model. We mainly use relational data model in our concepts.
- Data model is a conceptual tool to describe data, relationships among data, semantics of data and consistency constraints of the data.
- Relational data model uses a set of tables or relations , each of which is assigned a unique name to represent both data and the relationships among those data.



Formal Relational Term	Informal Equivalence
Relation	Table
Tuple	Row or Record
Cardinality of a Relation	Number of rows
Attribute	Column or Field
Degree of a Relation	Number of Columns
Primary Key	Unique Identifier
Domain	A pool of values from which the value of specific attributes of specific relations are taken

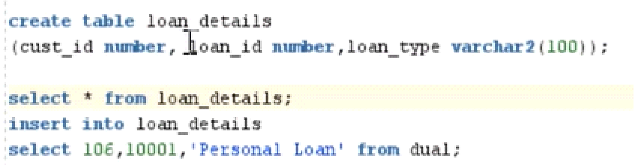
Sample syntax of Oracle:



```
select * from customer_details;

desc customer_details;
/

insert into customer_details (cust_id,cust_last_name)
values (101,'Virat');
```

Line 1: To print all lines of customer_details table
Line 2: To describe the structure of customer_details table
Line 3: To insert a value column specifically into the table

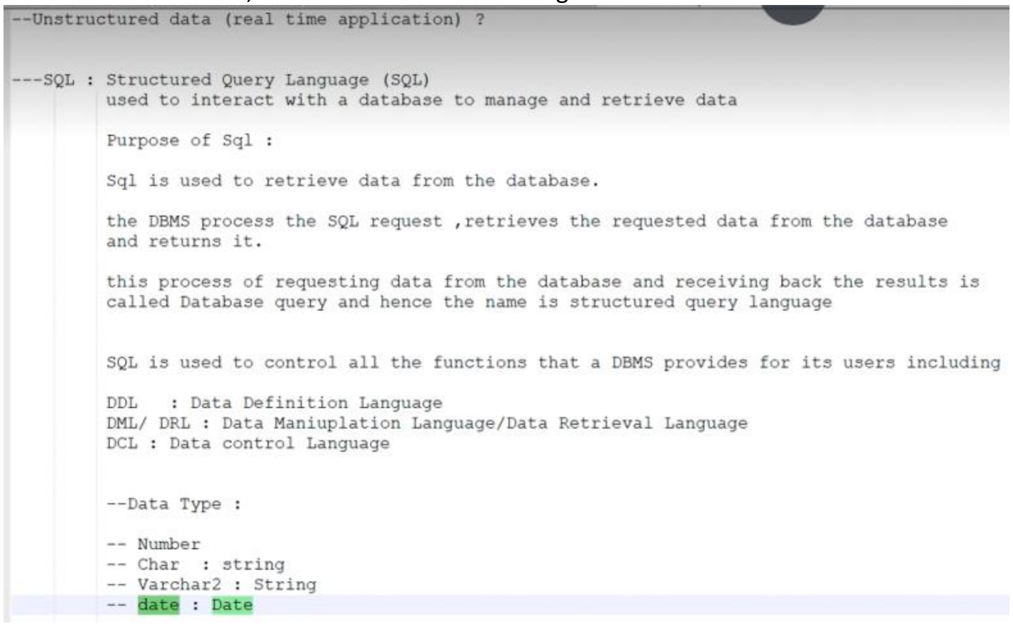
- For Relational data model first we need to mention or have data model for the database we have. So we can establish relations among the data tables.
- Group of unique key's can be declared as primary key but it is actually called candidate key.
- If any field is empty it is not "null" string or empty space character it is called undefined.



```
create table loan_details
(cust_id number, loan_id number,loan_type varchar2(100));

select * from loan_details;
insert into loan_details
select 106,10001,'Personal Loan' from dual;
```

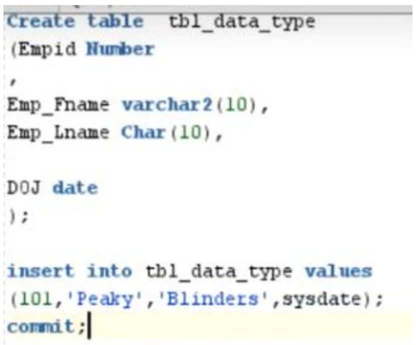
- Line 1: Creating a table
- Inserting into new table by copying from other table or just doing sample copy new data from dual.
- If we create cust_id of loan details as the foreign key of cust_details then if we enter the following line 2 command we would get an error because 106 cust id is not in table 1 customer_details.
- We can customize error's we can define our own errors in oracle.
- Relational Database is any database in which the data is logically organized based on the

relational model, RDMS is DBMS which manages the relational model.



```
--Unstructured data (real time application) ?

---SQL : Structured Query Language (SQL)
         used to interact with a database to manage and retrieve data

         Purpose of Sql :

         Sql is used to retrieve data from the database.

         the DBMS process the SQL request ,retrieves the requested data from the database
         and returns it.

         this process of requesting data from the database and receiving back the results is
         called Database query and hence the name is structured query language


         SQL is used to control all the functions that a DBMS provides for its users including

         DDL   : Data Definition Language
         DML/ DRL : Data Maniuplation Language/Data Retrieval Language
         DCL : Data control Language


         --Data Type :

         -- Number
         -- Char   : string
         -- Varchar2 : String
         -- date : Date
```

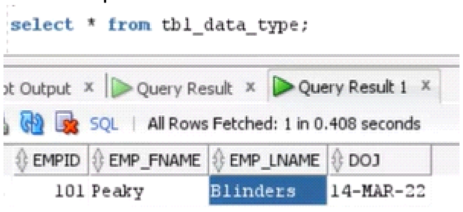
How to create a table in the database and how to insert the data into database



```
Create table  tbl_data_type
(Empid Number
,
Emp_Fname varchar2(10),
Emp_Lname Char(10),

DOJ date
);

insert into tbl_data_type values
(101,'Peaky','Blinders',sysdate);
commit;
```

We need to commit by ourself because the values will be updated only on our computer if we are not committing but if we commit the data is updated for the whole team. So we need to remember to commit the database to ensure the uniformity among the team.

- To print the data out of the table

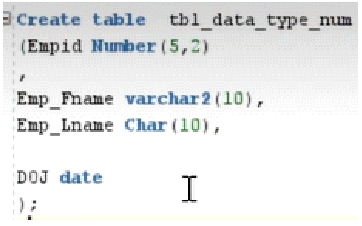


```
select * from tbl_data_type;
```

EMPID	EMP_FNAME	EMP_LNAME	DOJ
101	Peaky	Blinders	14-MAR-22

Char takes memory for how much we defined. Varchar2 takes the space how much we occupied until the limit we declared.

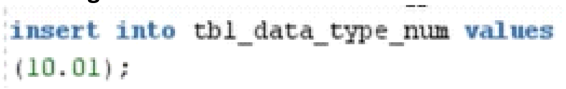
Creating a table with a column of decimal values and a date column



```
Create table  tbl_data_type_num
(Empid Number(5,2)
,
Emp_Fname varchar2(10),
Emp_Lname Char(10),

DOJ date
);
```

- Here we are going to have employee id of total 5 digits 3 digits at the front and 2 digits as the decimal

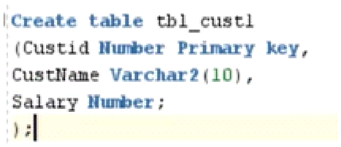
Inserting into such decimal valued table



```
insert into tbl_data_type_num values
(10.01);
```

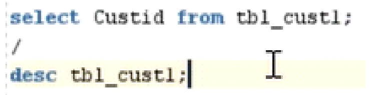
- Dropping the table makes us to dropping the data with it's structure.
- Truncate makes to drop the content of the table but saving table structure.

Syntax to create a table



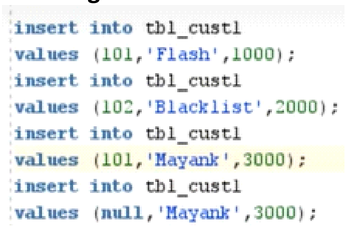
```sql
Create table tbl_custl
(Custid Number Primary key,
CustName Varchar2(10),
Salary Number;
);
```

Printing the data of the table and getting the structure of the database using desc keyword.



```sql
select Custid from tbl_custl;
/
desc tbl_custl;
```

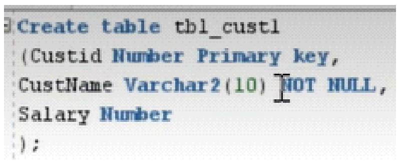
Inserting into the table



```sql
insert into tbl_custl
values (101,'Flash',1000);
insert into tbl_custl
values (102,'Blacklist',2000);
insert into tbl_custl
values (101,'Mayank',3000);
insert into tbl_custl
values (null,'Mayank',3000);
```

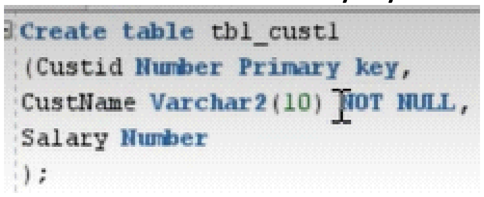
In primary key we must not have duplicate value and null values.

- We can add some constraints on the attributes

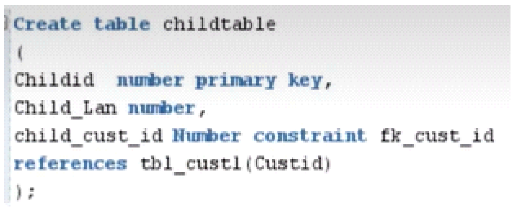


```sql
Create table tbl_custl
(Custid Number Primary key,
CustName Varchar2(10) NOT NULL,
Salary Number
);
```

Primary Key and Foreign Key



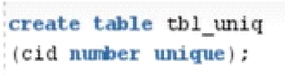
```sql
Create table tbl_custl
(Custid Number Primary key,
CustName Varchar2(10) NOT NULL,
Salary Number
);
```



```sql
Create table childtable
(
Childid  number primary key,
Child_Lan number,
child_cust_id Number constraint fk_cust_id
references tbl_custl(Custid)
);
```

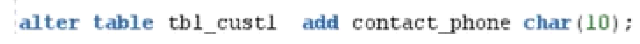
- Here we are creating an attribute of a table as a foreign key of an attribute of another table where it is declared as primary key.
- Foreign key is referencing the primary key.
- Every data of foreign key must be present in primary key. But not vice versa. If we give some data that we entered in foreign key that is not there in primary key then we would get error. But we can add data that is not there in foreign key.

Unique Key Constraint

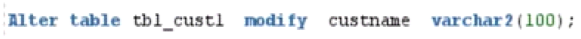


```sql
create table tbl_uniq
(cid number unique);
```

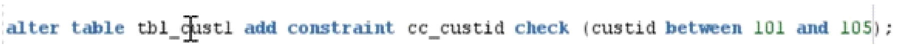
Alter command



```sql
alter table tbl_custl  add contact_phone char(10);
```



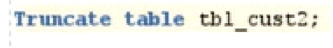
```sql
Alter table tbl_custl  modify  custname  varchar2(100);
```



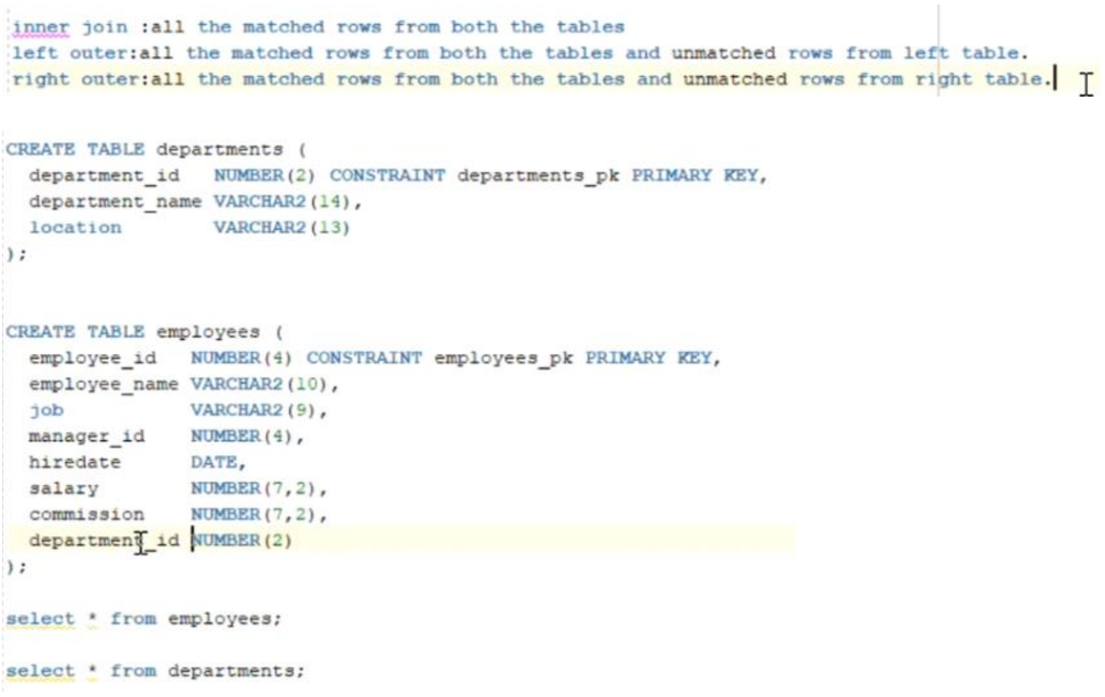
```sql
alter table tbl_custl add constraint cc_custid check (custid between 101 and 105);
```

While removing tables using drop we must follow some rules for example if we want to remove any parent table , first of all we must remove the child tables and then we need to remove the parent table. Because child table is dependent on parent's table attribute.

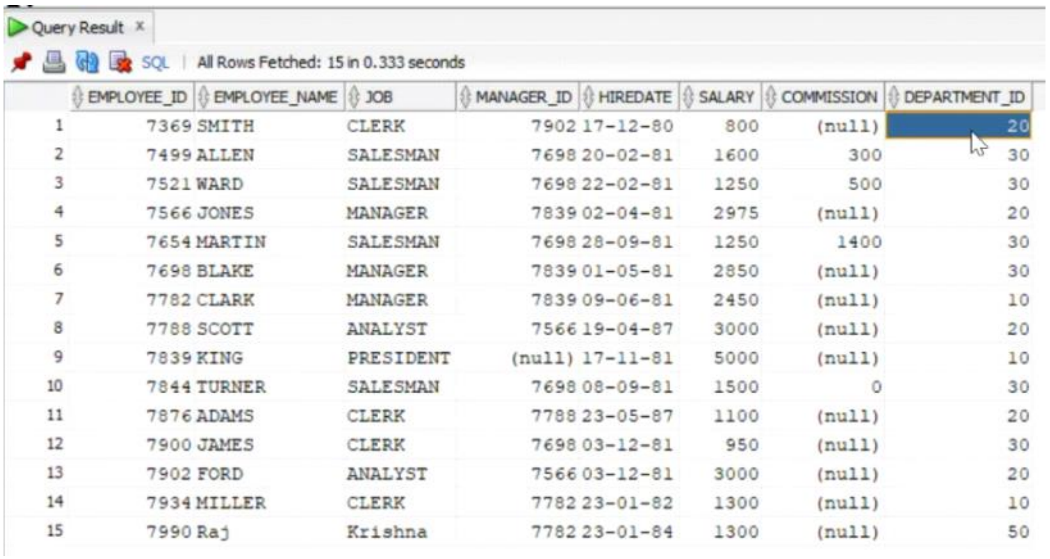
Truncate the table:



```sql
Truncate table tbl_cust2;
```



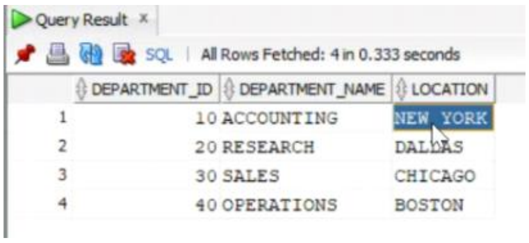
```
inner join :all the matched rows from both the tables
left outer:all the matched rows from both the tables and unmatched rows from left table.
right outer:all the matched rows from both the tables and unmatched rows from right table.


CREATE TABLE departments (
  department_id   NUMBER(2) CONSTRAINT departments_pk PRIMARY KEY,
  department_name VARCHAR2(14),
  location        VARCHAR2(13)
);


CREATE TABLE employees (
  employee_id   NUMBER(4) CONSTRAINT employees_pk PRIMARY KEY,
  employee_name VARCHAR2(10),
  job           VARCHAR2(9),
  manager_id    NUMBER(4),
  hiredate      DATE,
  salary        NUMBER(7,2),
  commission    NUMBER(7,2),
  department_id NUMBER(2)
);

select * from employees;

select * from departments;
```



Query Result ×

SQL | All Rows Fetched: 15 in 0.333 seconds

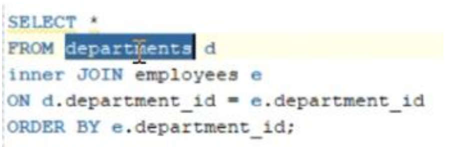
	EMPLOYEE_ID	EMPLOYEE_NAME	JOB	MANAGER_ID	HIREDATE	SALARY	COMMISSION	DEPARTMENT_ID
1	7369	SMITH	CLERK	7902	17-12-80	800	(null)	20
2	7499	ALLEN	SALESMAN	7698	20-02-81	1600	300	30
3	7521	WARD	SALESMAN	7698	22-02-81	1250	500	30
4	7566	JONES	MANAGER	7839	02-04-81	2975	(null)	20
5	7654	MARTIN	SALESMAN	7698	28-09-81	1250	1400	30
6	7698	BLAKE	MANAGER	7839	01-05-81	2850	(null)	30
7	7782	CLARK	MANAGER	7839	09-06-81	2450	(null)	10
8	7788	SCOTT	ANALYST	7566	19-04-87	3000	(null)	20
9	7839	KING	PRESIDENT	(null)	17-11-81	5000	(null)	10
10	7844	TURNER	SALESMAN	7698	08-09-81	1500	0	30
11	7876	ADAMS	CLERK	7788	23-05-87	1100	(null)	20
12	7900	JAMES	CLERK	7698	03-12-81	950	(null)	30
13	7902	FORD	ANALYST	7566	03-12-81	3000	(null)	20
14	7934	MILLER	CLERK	7782	23-01-82	1300	(null)	10
15	7990	Raj	Krishna	7782	23-01-84	1300	(null)	50



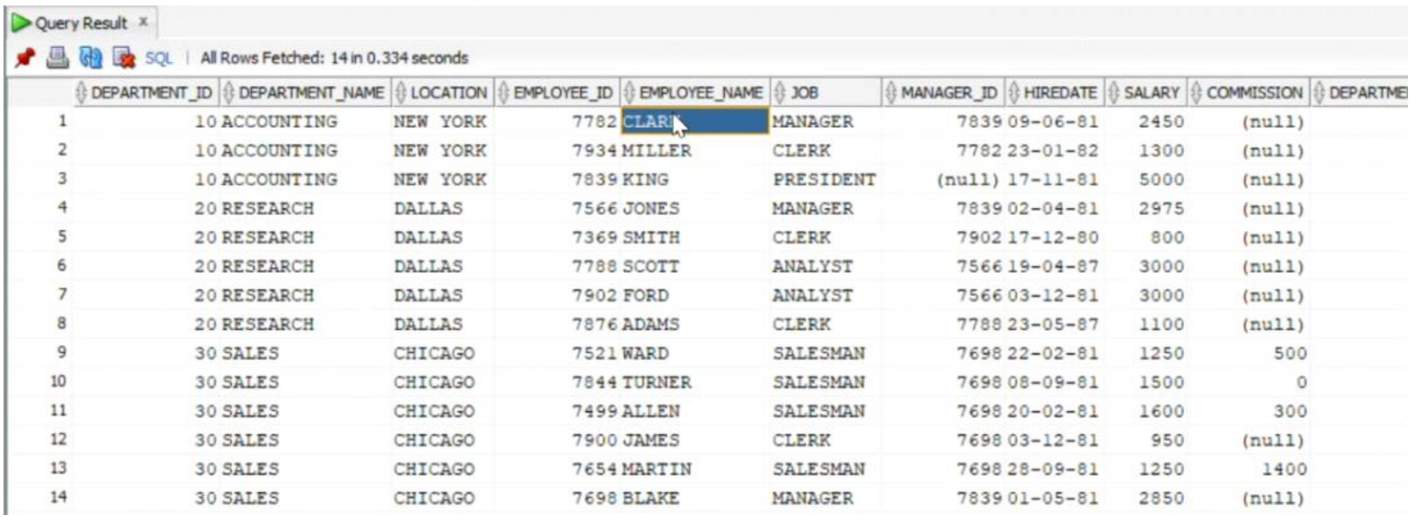
Query Result ×

SQL | All Rows Fetched: 4 in 0.333 seconds

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON



```
SELECT *
FROM departments d
inner JOIN employees e
ON d.department_id = e.department_id
ORDER BY e.department_id;
```

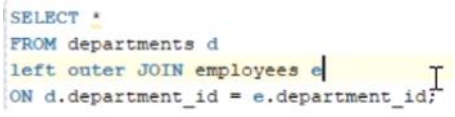


Query Result ×

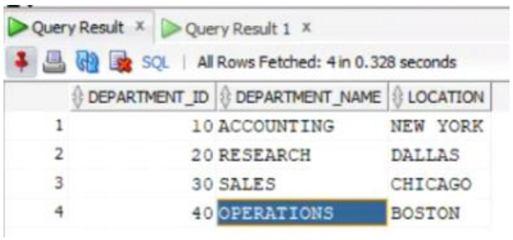
SQL | All Rows Fetched: 14 in 0.334 seconds

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION	EMPLOYEE_ID	EMPLOYEE_NAME	JOB	MANAGER_ID	HIREDATE	SALARY	COMMISSION	DEPARTME
1	10	ACCOUNTING	NEW YORK	7782	CLARK	MANAGER	7839	09-06-81	2450	(null)	
2	10	ACCOUNTING	NEW YORK	7934	MILLER	CLERK	7782	23-01-82	1300	(null)	
3	10	ACCOUNTING	NEW YORK	7839	KING	PRESIDENT	(null)	17-11-81	5000	(null)	
4	20	RESEARCH	DALLAS	7566	JONES	MANAGER	7839	02-04-81	2975	(null)	
5	20	RESEARCH	DALLAS	7369	SMITH	CLERK	7902	17-12-80	800	(null)	
6	20	RESEARCH	DALLAS	7788	SCOTT	ANALYST	7566	19-04-87	3000	(null)	
7	20	RESEARCH	DALLAS	7902	FORD	ANALYST	7566	03-12-81	3000	(null)	
8	20	RESEARCH	DALLAS	7876	ADAMS	CLERK	7788	23-05-87	1100	(null)	
9	30	SALES	CHICAGO	7521	WARD	SALESMAN	7698	22-02-81	1250	500	
10	30	SALES	CHICAGO	7844	TURNER	SALESMAN	7698	08-09-81	1500	0	
11	30	SALES	CHICAGO	7499	ALLEN	SALESMAN	7698	20-02-81	1600	300	
12	30	SALES	CHICAGO	7900	JAMES	CLERK	7698	03-12-81	950	(null)	
13	30	SALES	CHICAGO	7654	MARTIN	SALESMAN	7698	28-09-81	1250	1400	
14	30	SALES	CHICAGO	7698	BLAKE	MANAGER	7839	01-05-81	2850	(null)	

Oracle Thumb rule : If we have n tables we can make almost n-1 joins



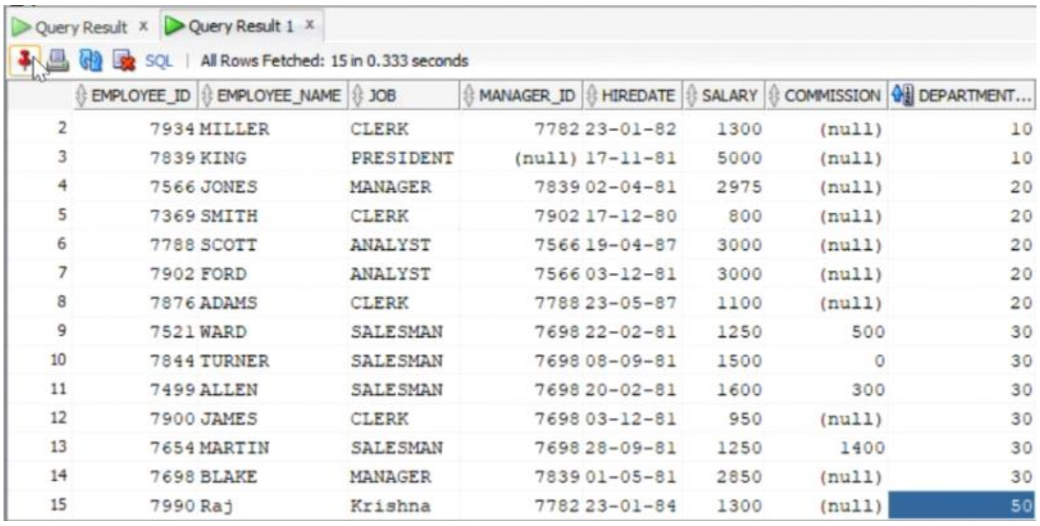
```
SELECT *
FROM departments d
left outer JOIN employees e
ON d.department_id = e.department_id;
```



Query Result × Query Result 1 ×

SQL | All Rows Fetched: 4 in 0.328 seconds

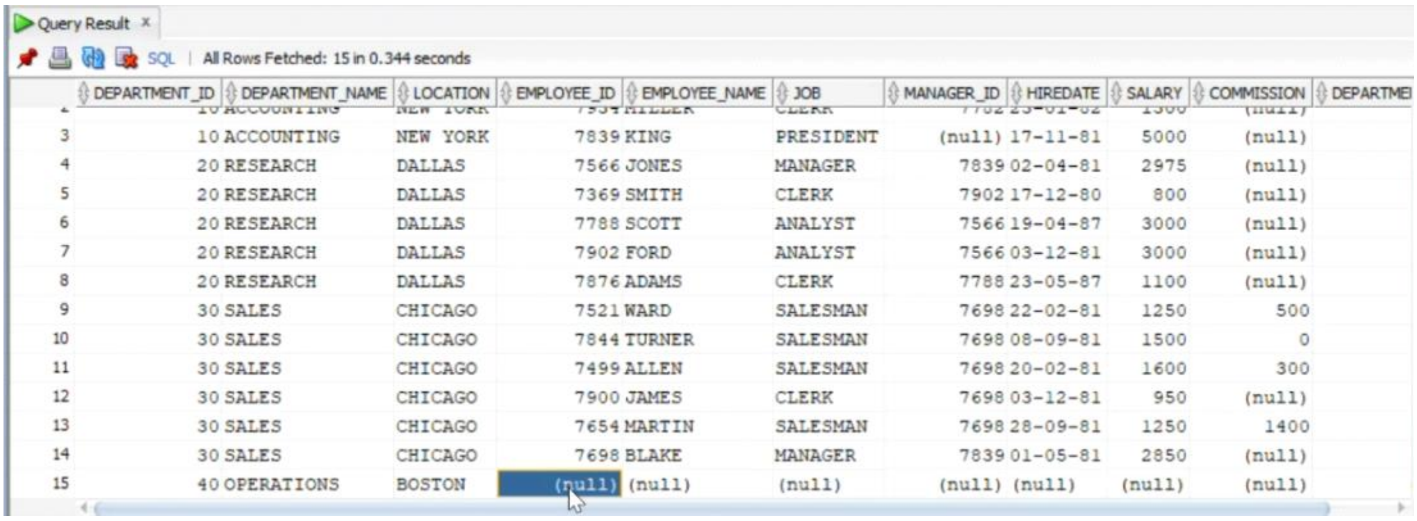
	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON



Query Result × Query Result 1 ×

SQL | All Rows Fetched: 15 in 0.333 seconds

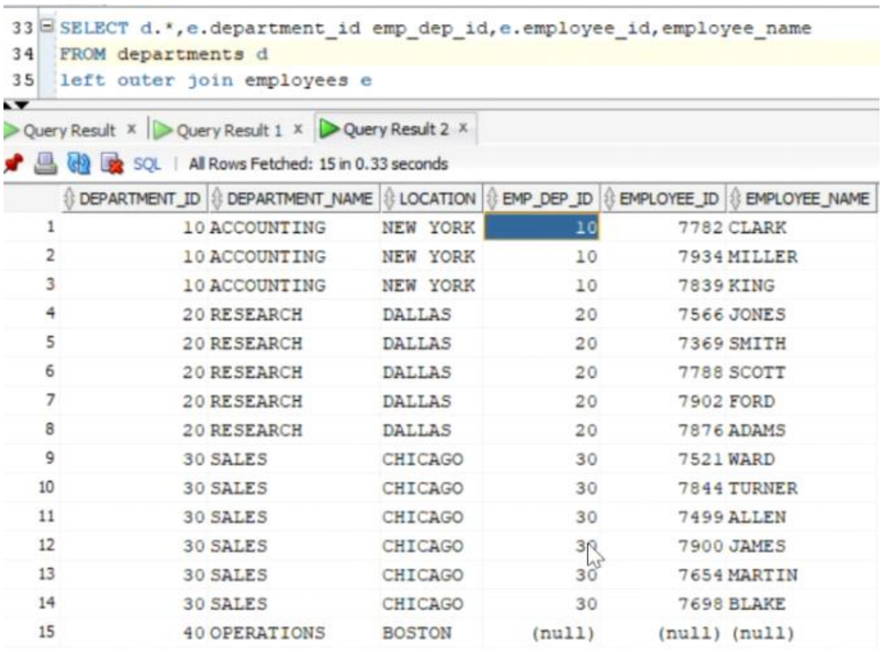
	EMPLOYEE_ID	EMPLOYEE_NAME	JOB	MANAGER_ID	HIREDATE	SALARY	COMMISSION	DEPARTMENT...
2	7934	MILLER	CLERK	7782	23-01-82	1300	(null)	10
3	7839	KING	PRESIDENT	(null)	17-11-81	5000	(null)	10
4	7566	JONES	MANAGER	7839	02-04-81	2975	(null)	20
5	7369	SMITH	CLERK	7902	17-12-80	800	(null)	20
6	7788	SCOTT	ANALYST	7566	19-04-87	3000	(null)	20
7	7902	FORD	ANALYST	7566	03-12-81	3000	(null)	20
8	7876	ADAMS	CLERK	7788	23-05-87	1100	(null)	20
9	7521	WARD	SALESMAN	7698	22-02-81	1250	500	30
10	7844	TURNER	SALESMAN	7698	08-09-81	1500	0	30
11	7499	ALLEN	SALESMAN	7698	20-02-81	1600	300	30
12	7900	JAMES	CLERK	7698	03-12-81	950	(null)	30
13	7654	MARTIN	SALESMAN	7698	28-09-81	1250	1400	30
14	7698	BLAKE	MANAGER	7839	01-05-81	2850	(null)	30
15	7990	Raj	Krishna	7782	23-01-84	1300	(null)	50



Query Result ×

SQL | All Rows Fetched: 15 in 0.344 seconds

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION	EMPLOYEE_ID	EMPLOYEE_NAME	JOB	MANAGER_ID	HIREDATE	SALARY	COMMISSION	DEPARTME
3	10	ACCOUNTING	NEW YORK	7839	KING	PRESIDENT	(null)	17-11-81	5000	(null)	
4	20	RESEARCH	DALLAS	7566	JONES	MANAGER	7839	02-04-81	2975	(null)	
5	20	RESEARCH	DALLAS	7369	SMITH	CLERK	7902	17-12-80	800	(null)	
6	20	RESEARCH	DALLAS	7788	SCOTT	ANALYST	7566	19-04-87	3000	(null)	
7	20	RESEARCH	DALLAS	7902	FORD	ANALYST	7566	03-12-81	3000	(null)	
8	20	RESEARCH	DALLAS	7876	ADAMS	CLERK	7788	23-05-87	1100	(null)	
9	30	SALES	CHICAGO	7521	WARD	SALESMAN	7698	22-02-81	1250	500	
10	30	SALES	CHICAGO	7844	TURNER	SALESMAN	7698	08-09-81	1500	0	
11	30	SALES	CHICAGO	7499	ALLEN	SALESMAN	7698	20-02-81	1600	300	
12	30	SALES	CHICAGO	7900	JAMES	CLERK	7698	03-12-81	950	(null)	
13	30	SALES	CHICAGO	7654	MARTIN	SALESMAN	7698	28-09-81	1250	1400	
14	30	SALES	CHICAGO	7698	BLAKE	MANAGER	7839	01-05-81	2850	(null)	
15	40	OPERATIONS	BOSTON	(null)	(null)	(null)	(null)	(null)	(null)	(null)	

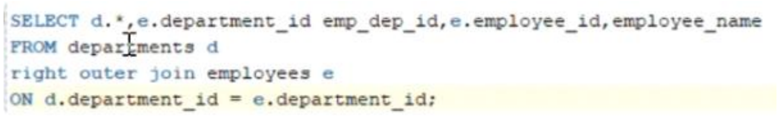


```sql
SELECT d.*,e.department_id emp_dep_id,e.employee_id,employee_name
FROM departments d
left outer join employees e
```

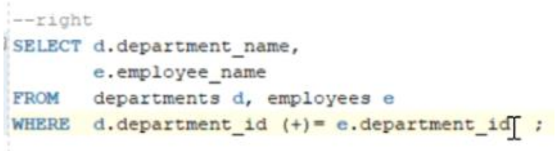
Query Result × | Query Result 1 × | Query Result 2 ×

SQL | All Rows Fetched: 15 in 0.33 seconds

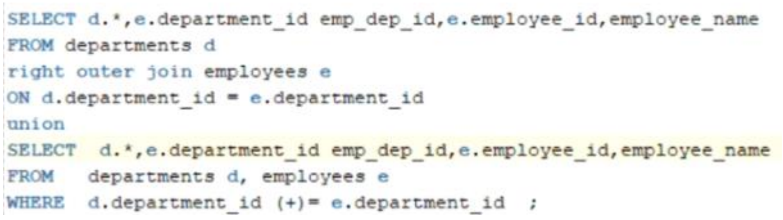
	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION	EMP_DEP_ID	EMPLOYEE_ID	EMPLOYEE_NAME
1	10	ACCOUNTING	NEW YORK	10	7782	CLARK
2	10	ACCOUNTING	NEW YORK	10	7934	MILLER
3	10	ACCOUNTING	NEW YORK	10	7839	KING
4	20	RESEARCH	DALLAS	20	7566	JONES
5	20	RESEARCH	DALLAS	20	7369	SMITH
6	20	RESEARCH	DALLAS	20	7788	SCOTT
7	20	RESEARCH	DALLAS	20	7902	FORD
8	20	RESEARCH	DALLAS	20	7876	ADAMS
9	30	SALES	CHICAGO	30	7521	WARD
10	30	SALES	CHICAGO	30	7844	TURNER
11	30	SALES	CHICAGO	30	7499	ALLEN
12	30	SALES	CHICAGO	30	7900	JAMES
13	30	SALES	CHICAGO	30	7654	MARTIN
14	30	SALES	CHICAGO	30	7698	BLAKE
15	40	OPERATIONS	BOSTON	(null)	(null)	(null)



```sql
SELECT d.*,e.department_id emp_dep_id,e.employee_id,employee_name
FROM departments d
right outer join employees e
ON d.department_id = e.department_id;
```

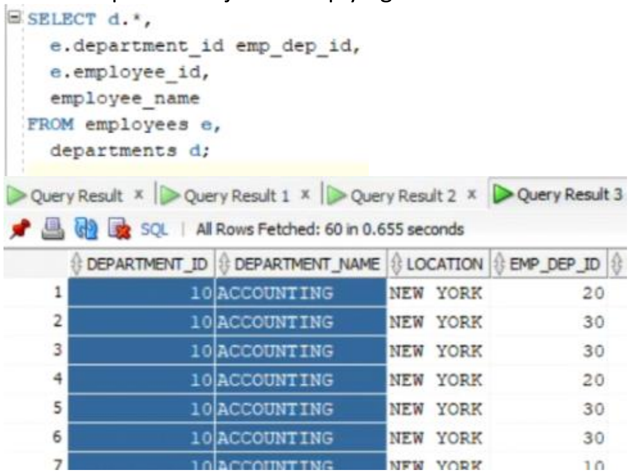


```sql
--right
SELECT d.department_name,
       e.employee_name
FROM   departments d, employees e
WHERE  d.department_id (+)= e.department_id ;
```



```sql
SELECT d.*,e.department_id emp_dep_id,e.employee_id,employee_name
FROM departments d
right outer join employees e
ON d.department_id = e.department_id
union
SELECT  d.*,e.department_id emp_dep_id,e.employee_id,employee_name
FROM   departments d, employees e
WHERE  d.department_id (+)= e.department_id  ;
```

Cartesian product is just multiplying the rows of table A to table B



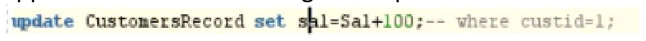
```sql
SELECT d.*,
  e.department_id emp_dep_id,
  e.employee_id,
  employee_name
FROM employees e,
  departments d;
```

Query Result × | Query Result 1 × | Query Result 2 × | Query Result 3

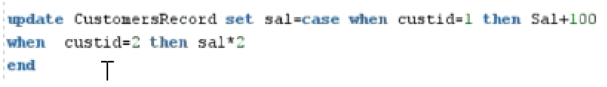
SQL | All Rows Fetched: 60 in 0.655 seconds

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION	EMP_DEP_ID
1	10	ACCOUNTING	NEW YORK	20
2	10	ACCOUNTING	NEW YORK	30
3	10	ACCOUNTING	NEW YORK	30
4	10	ACCOUNTING	NEW YORK	20
5	10	ACCOUNTING	NEW YORK	30
6	10	ACCOUNTING	NEW YORK	30
7	10	ACCOUNTING	NEW YORK	10

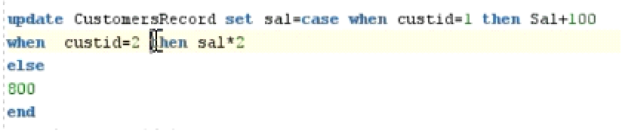
Cartesian product filtering: First do filtering and does cartesian product , SQL does bottom up approch while we are having where predicate.



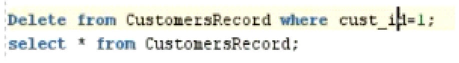
```sql
update CustomersRecord set sal=Sal+100;-- where custid=1;
```



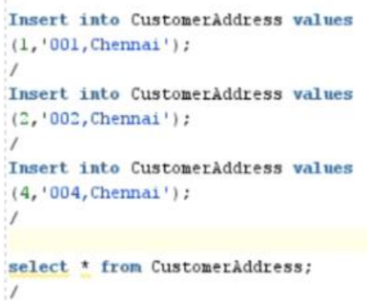
```sql
update CustomersRecord set sal=case when custid=1 then Sal+100
when  custid=2 then sal*2
end
```



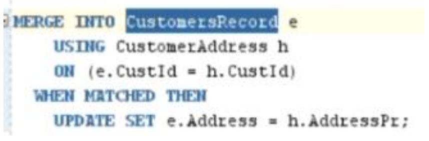
```
update CustomersRecord set sal=case when custid=1 then Sal+100
when  custid=2 then sal*2
else
800
end
```



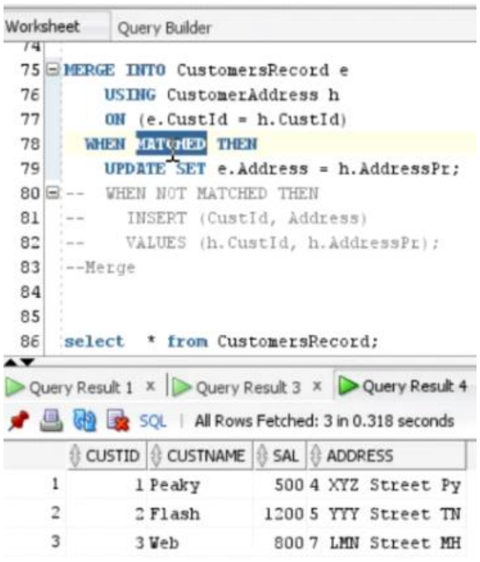
```
Delete from CustomersRecord where cust_id=1;
select * from CustomersRecord;
```



```
Insert into CustomerAddress values
(1,'001,Chennai');
/
Insert into CustomerAddress values
(2,'002,Chennai');
/
Insert into CustomerAddress values
(4,'004,Chennai');
/

select * from CustomerAddress;
/
```



```
MERGE INTO CustomersRecord e
    USING CustomerAddress h
    ON (e.CustId = h.CustId)
  WHEN MATCHED THEN
    UPDATE SET e.Address = h.AddressPr;
```



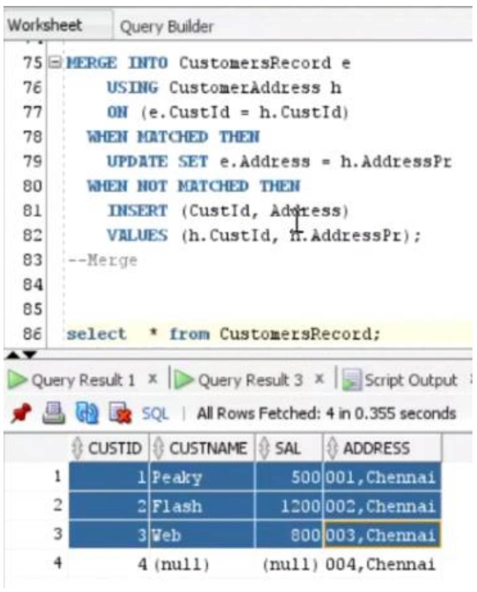
Worksheet | Query Builder

```
MERGE INTO CustomersRecord e
    USING CustomerAddress h
    ON (e.CustId = h.CustId)
  WHEN MATCHED THEN
    UPDATE SET e.Address = h.AddressPr;
--  WHEN NOT MATCHED THEN
--    INSERT (CustId, Address)
--    VALUES (h.CustId, h.AddressPr);
--Merge


select  * from CustomersRecord;
```

Query Result 1 × | Query Result 3 × | Query Result 4

SQL | All Rows Fetched: 3 in 0.318 seconds

	CUSTID	CUSTNAME	SAL	ADDRESS
1	1	Peaky	500	4 XYZ Street Py
2	2	Flash	1200	5 YYY Street TN
3	3	Web	800	7 LMN Street MH



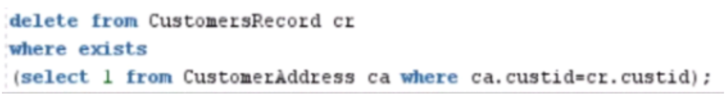
Worksheet | Query Builder

```
MERGE INTO CustomersRecord e
    USING CustomerAddress h
    ON (e.CustId = h.CustId)
  WHEN MATCHED THEN
    UPDATE SET e.Address = h.AddressPr
  WHEN NOT MATCHED THEN
    INSERT (CustId, Address)
    VALUES (h.CustId, h.AddressPr);
--Merge


select  * from CustomersRecord;
```

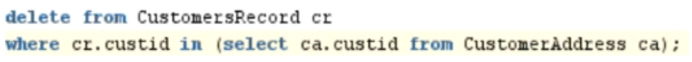
Query Result 1 × | Query Result 3 × | Script Output

SQL | All Rows Fetched: 4 in 0.355 seconds

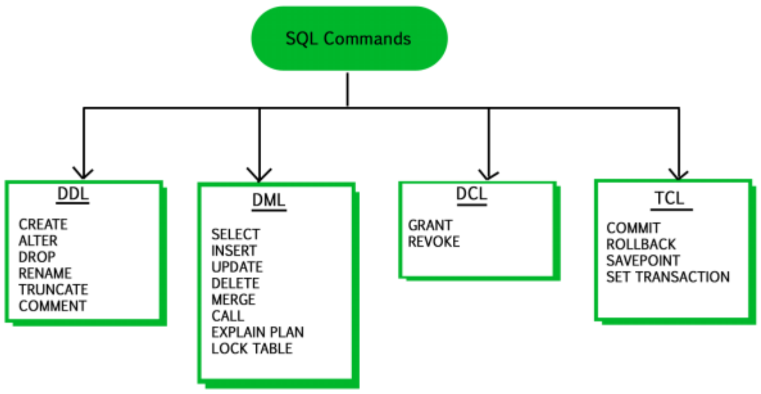
	CUSTID	CUSTNAME	SAL	ADDRESS
1	1	Peaky	500	001,Chennai
2	2	Flash	1200	002,Chennai
3	3	Web	800	003,Chennai
4	4	(null)	(null)	004,Chennai

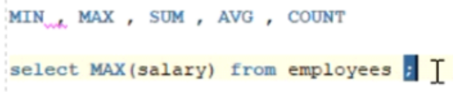


```
delete from CustomersRecord cr
where exists
(select 1 from CustomerAddress ca where ca.custid=cr.custid);
```

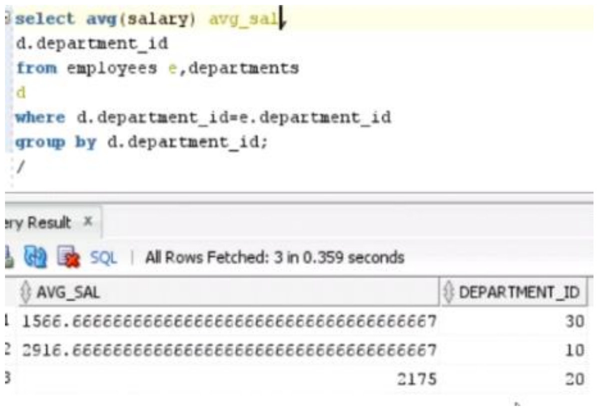


```
delete from CustomersRecord cr
where cr.custid in (select ca.custid from CustomerAddress ca);
```



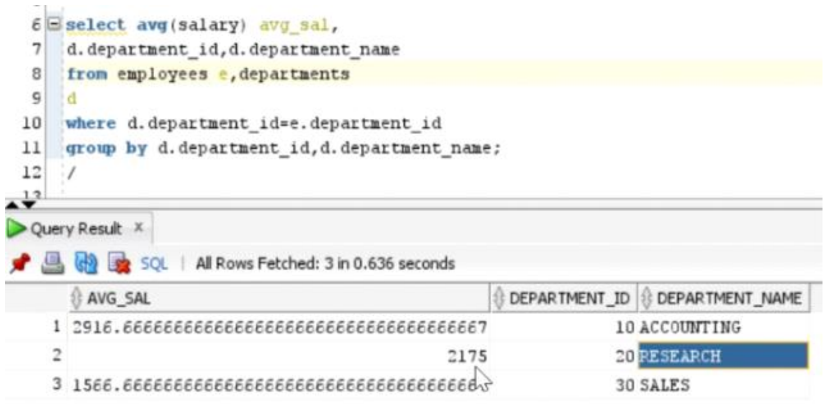


```
MIN , MAX , SUM , AVG , COUNT

select MAX(salary) from employees ;
```



```
select avg(salary) avg_sal,
d.department_id
from employees e,departments
d
where d.department_id=e.department_id
group by d.department_id;
/
```

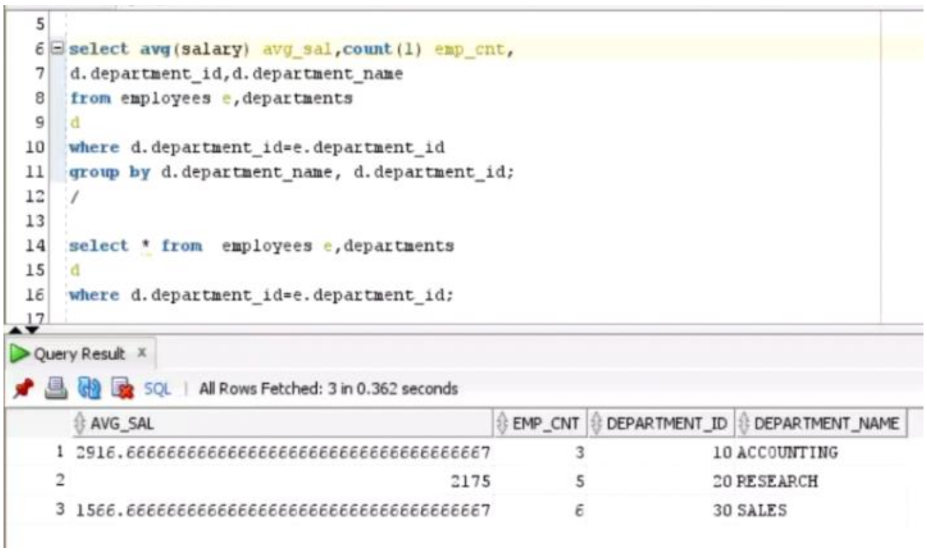
```
ry Result ×
SQL | All Rows Fetched: 3 in 0.359 seconds

  AVG_SAL                                 DEPARTMENT_ID
1 1566.6666666666666666666666666666666667            30
2 2916.6666666666666666666666666666666667            10
3                                    2175            20
```



```
select avg(salary) avg_sal,
d.department_id,d.department_name
from employees e,departments
d
where d.department_id=e.department_id
group by d.department_id,d.department_name;
/
```

```
Query Result ×
SQL | All Rows Fetched: 3 in 0.636 seconds

  AVG_SAL                                 DEPARTMENT_ID DEPARTMENT_NAME
1 2916.6666666666666666666666666666666667            10 ACCOUNTING
2                                    2175            20 RESEARCH
3 1566.6666666666666666666666666666666667            30 SALES
```

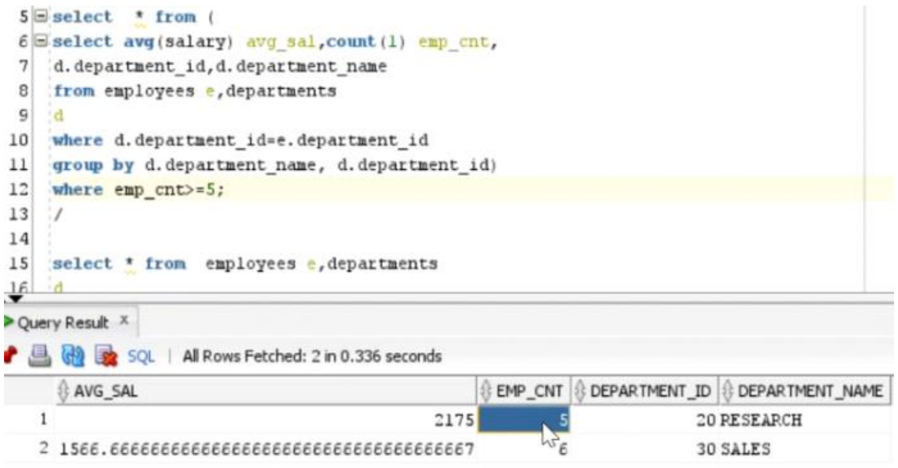
If we use aggregate statement in the select statement singly then there will be no issue. But if we try to print aggregate statement with some other columns to print we must add group class as above.



```
select avg(salary) avg_sal,count(1) emp_cnt,
d.department_id,d.department_name
from employees e,departments
d
where d.department_id=e.department_id
group by d.department_name, d.department_id;
/

select * from  employees e,departments
d
where d.department_id=e.department_id;
```

All Rows Fetched: 3 in 0.362 seconds

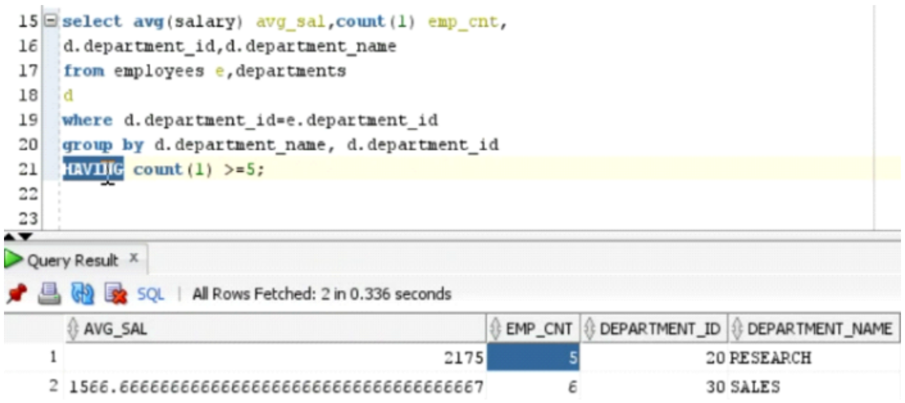
```
  AVG_SAL                                      EMP_CNT  DEPARTMENT_ID  DEPARTMENT_NAME
1 2916.666666666666666666666666666666666667          3             10  ACCOUNTING
2                                        2175        5             20  RESEARCH
3 1566.666666666666666666666666666666666667          6             30  SALES
```



```
select  * from (
select avg(salary) avg_sal,count(1) emp_cnt,
d.department_id,d.department_name
from employees e,departments
d
where d.department_id=e.department_id
group by d.department_name, d.department_id)
where emp_cnt>=5;
/

select * from  employees e,departments
d
```

All Rows Fetched: 2 in 0.336 seconds

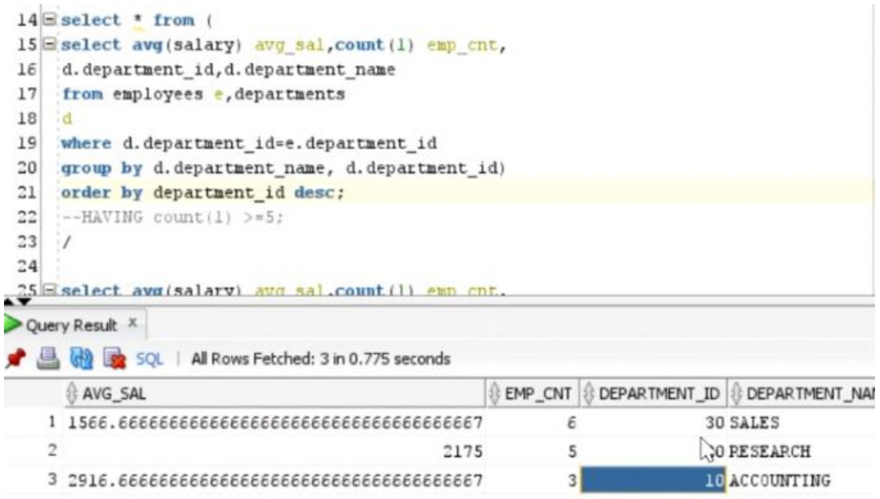
```
  AVG_SAL                                      EMP_CNT  DEPARTMENT_ID  DEPARTMENT_NAME
1                                        2175        5             20  RESEARCH
2 1566.666666666666666666666666666666666667          6             30  SALES
```



```
select avg(salary) avg_sal,count(1) emp_cnt,
d.department_id,d.department_name
from employees e,departments
d
where d.department_id=e.department_id
group by d.department_name, d.department_id
HAVING count(1) >=5;
```

All Rows Fetched: 2 in 0.336 seconds

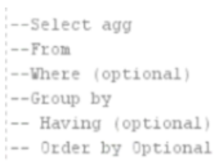
```
  AVG_SAL                                      EMP_CNT  DEPARTMENT_ID  DEPARTMENT_NAME
1                                        2175        5             20  RESEARCH
2 1566.666666666666666666666666666666666667          6             30  SALES
```



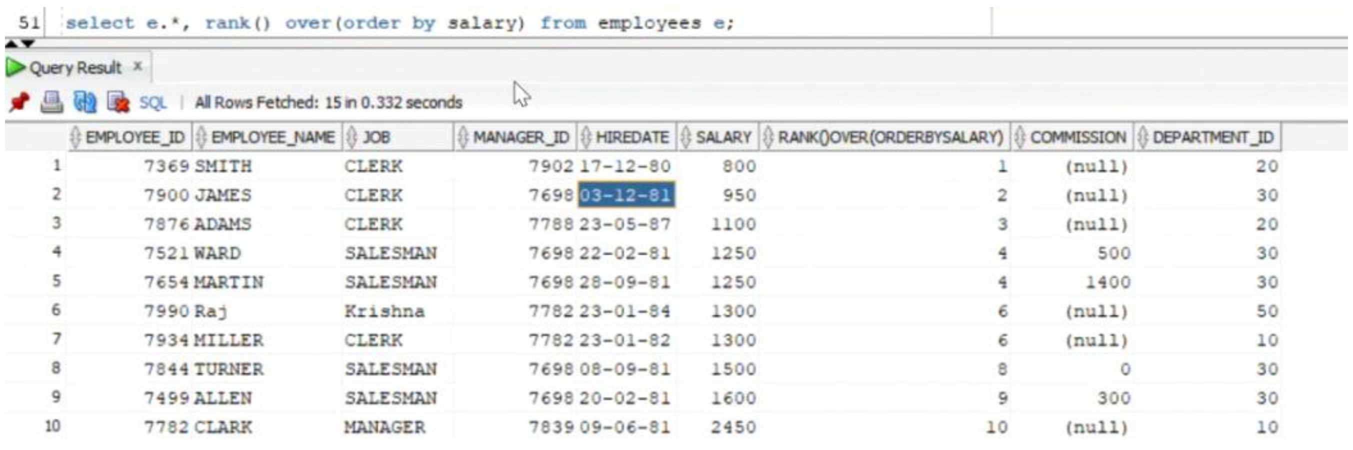
```
select * from (
select avg(salary) avg_sal,count(1) emp_cnt,
d.department_id,d.department_name
from employees e,departments
d
where d.department_id=e.department_id
group by d.department_name, d.department_id)
order by department_id desc;
--HAVING count(1) >=5;
/

select avg(salary) avg_sal,count(1) emp_cnt,
```

All Rows Fetched: 3 in 0.775 seconds

```
  AVG_SAL                                      EMP_CNT  DEPARTMENT_ID  DEPARTMENT_NAI
1 1566.666666666666666666666666666666666667          6             30  SALES
2                                        2175        5             20  RESEARCH
3 2916.666666666666666666666666666666666667          3             10  ACCOUNTING
```



```
--Select agg
--From
--Where (optional)
--Group by
-- Having (optional)
-- Order by Optional
```

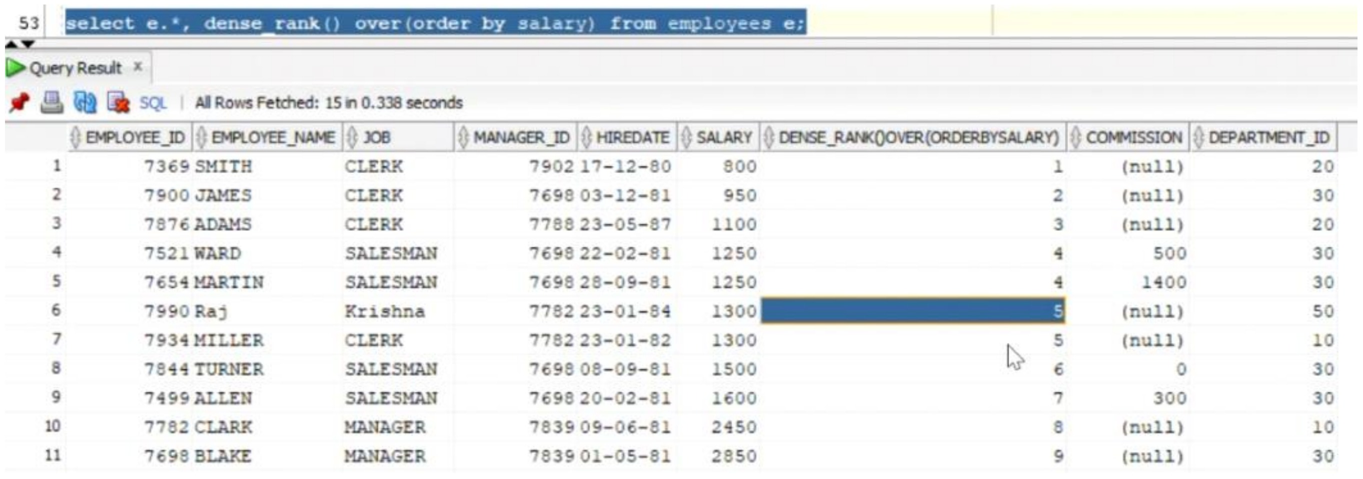


```
select e.*, rank() over(order by salary) from employees e;
```

Query Result

All Rows Fetched: 15 in 0.332 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	JOB	MANAGER_ID	HIREDATE	SALARY	RANK()OVER(ORDERBYSALARY)	COMMISSION	DEPARTMENT_ID
1	7369	SMITH	CLERK	7902	17-12-80	800	1	(null)	20
2	7900	JAMES	CLERK	7698	03-12-81	950	2	(null)	30
3	7876	ADAMS	CLERK	7788	23-05-87	1100	3	(null)	20
4	7521	WARD	SALESMAN	7698	22-02-81	1250	4	500	30
5	7654	MARTIN	SALESMAN	7698	28-09-81	1250	4	1400	30
6	7990	Raj	Krishna	7782	23-01-84	1300	6	(null)	50
7	7934	MILLER	CLERK	7782	23-01-82	1300	6	(null)	10
8	7844	TURNER	SALESMAN	7698	08-09-81	1500	8	0	30
9	7499	ALLEN	SALESMAN	7698	20-02-81	1600	9	300	30
10	7782	CLARK	MANAGER	7839	09-06-81	2450	10	(null)	10

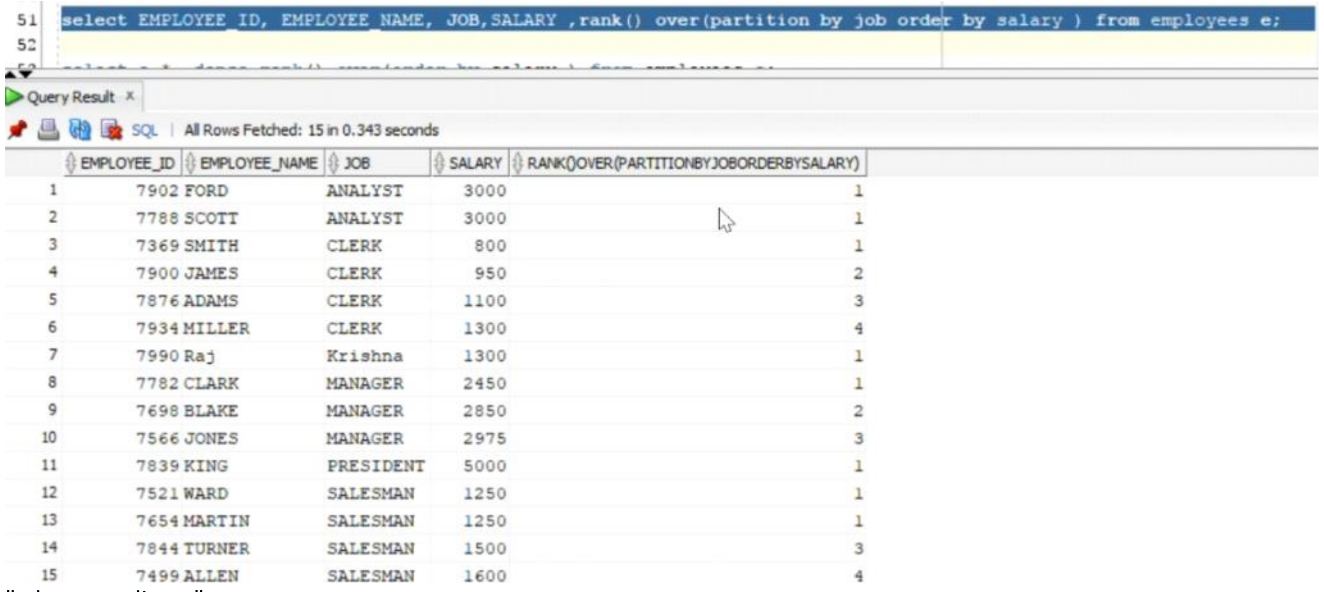


```
select e.*, dense_rank() over(order by salary) from employees e;
```

Query Result

All Rows Fetched: 15 in 0.338 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	JOB	MANAGER_ID	HIREDATE	SALARY	DENSE_RANK()OVER(ORDERBYSALARY)	COMMISSION	DEPARTMENT_ID
1	7369	SMITH	CLERK	7902	17-12-80	800	1	(null)	20
2	7900	JAMES	CLERK	7698	03-12-81	950	2	(null)	30
3	7876	ADAMS	CLERK	7788	23-05-87	1100	3	(null)	20
4	7521	WARD	SALESMAN	7698	22-02-81	1250	4	500	30
5	7654	MARTIN	SALESMAN	7698	28-09-81	1250	4	1400	30
6	7990	Raj	Krishna	7782	23-01-84	1300	5	(null)	50
7	7934	MILLER	CLERK	7782	23-01-82	1300	5	(null)	10
8	7844	TURNER	SALESMAN	7698	08-09-81	1500	6	0	30
9	7499	ALLEN	SALESMAN	7698	20-02-81	1600	7	300	30
10	7782	CLARK	MANAGER	7839	09-06-81	2450	8	(null)	10
11	7698	BLAKE	MANAGER	7839	01-05-81	2850	9	(null)	30



```
select EMPLOYEE_ID, EMPLOYEE_NAME, JOB,SALARY ,rank() over(partition by job order by salary ) from employees e;
```

Query Result

All Rows Fetched: 15 in 0.343 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	JOB	SALARY	RANK()OVER(PARTITIONBYJOBORDERBYSALARY)
1	7902	FORD	ANALYST	3000	1
2	7788	SCOTT	ANALYST	3000	1
3	7369	SMITH	CLERK	800	1
4	7900	JAMES	CLERK	950	2
5	7876	ADAMS	CLERK	1100	3
6	7934	MILLER	CLERK	1300	4
7	7990	Raj	Krishna	1300	1
8	7782	CLARK	MANAGER	2450	1
9	7698	BLAKE	MANAGER	2850	2
10	7566	JONES	MANAGER	2975	3
11	7839	KING	PRESIDENT	5000	1
12	7521	WARD	SALESMAN	1250	1
13	7654	MARTIN	SALESMAN	1250	1
14	7844	TURNER	SALESMAN	1500	3
15	7499	ALLEN	SALESMAN	1600	4

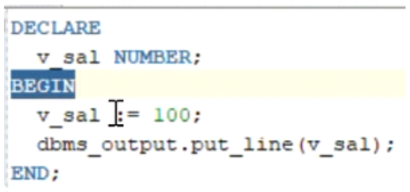
"where predicate"

Features of PL/SQL

1. PL/SQL is tightly integrated with SQL.
2. It offers extensive error checking.
3. It offers numerous data types.
4. It offers a variety of programming structures.
5. It supports structured programming through functions and procedures.
6. It supports the development of web applications and server pages.

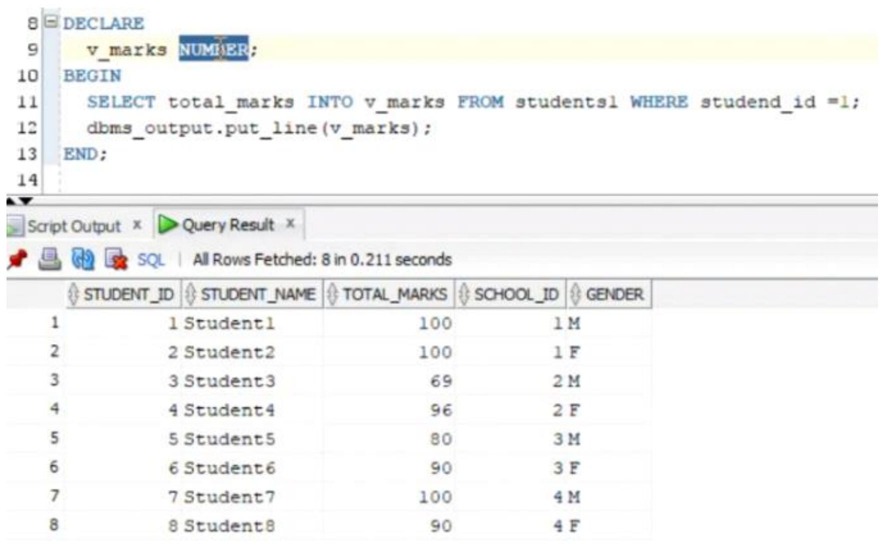
Advantages of PL/SQL

1. SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL.
2. PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
3. PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
4. PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
5. Applications written in PL/SQL are fully portable.
6. PL/SQL provides high security level.
7. PL/SQL provides access to predefined SQL packages.
8. PL/SQL provides support for developing Web Applications and Server Pages



```
DECLARE
  v_sal NUMBER;
BEGIN
  v_sal := 100;
  dbms_output.put_line(v_sal);
END;
```

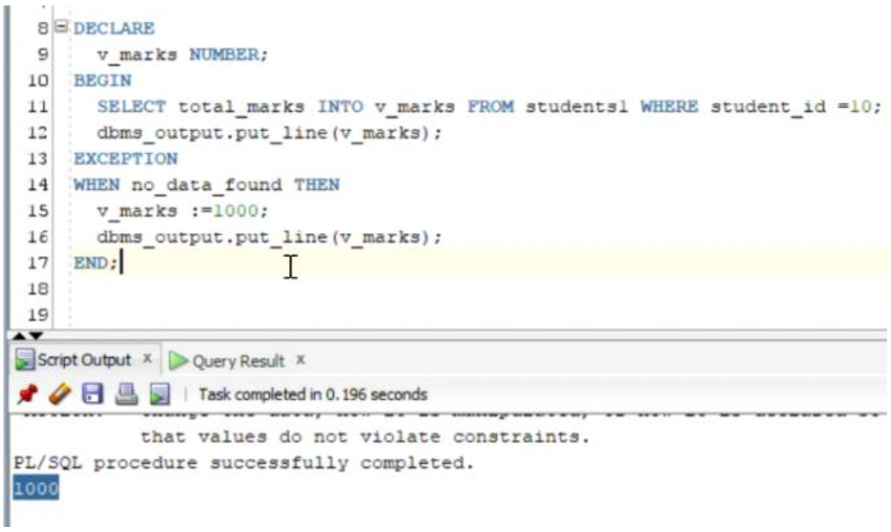
- The following dbms output put line is similar to console.log



```
DECLARE
  v_marks NUMBER;
BEGIN
  SELECT total_marks INTO v_marks FROM students1 WHERE studend_id =1;
  dbms_output.put_line(v_marks);
END;
```

-

	STUDENT_ID	STUDENT_NAME	TOTAL_MARKS	SCHOOL_ID	GENDER
1	1	Student1	100	1	M
2	2	Student2	100	1	F
3	3	Student3	69	2	M
4	4	Student4	96	2	F
5	5	Student5	80	3	M
6	6	Student6	90	3	F
7	7	Student7	100	4	M
8	8	Student8	90	4	F



```
DECLARE
  v_marks NUMBER;
BEGIN
  SELECT total_marks INTO v_marks FROM students1 WHERE student_id =10;
  dbms_output.put_line(v_marks);
EXCEPTION
WHEN no_data_found THEN
  v_marks :=1000;
  dbms_output.put_line(v_marks);
END;
```

```
          that values do not violate constraints.
PL/SQL procedure successfully completed.
1000
```

PL/SQL — Basic Syntax

DECLARE

<Declaration section>

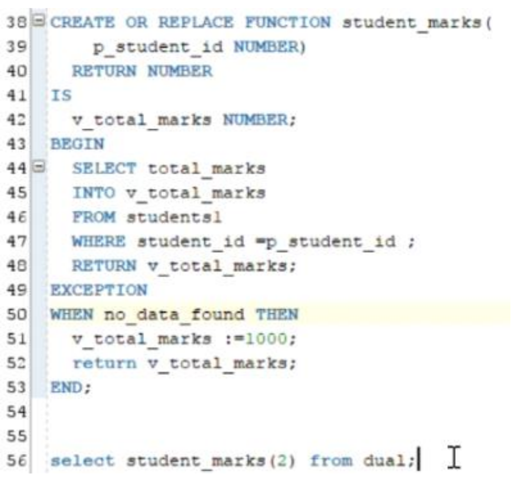
BEGIN

<Executable commands>

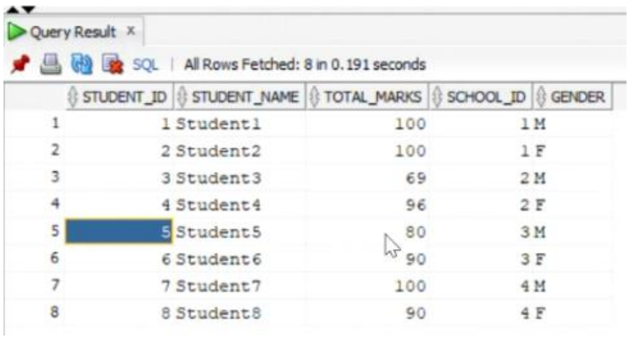
EXCEPTION

<Exception handling>

END;



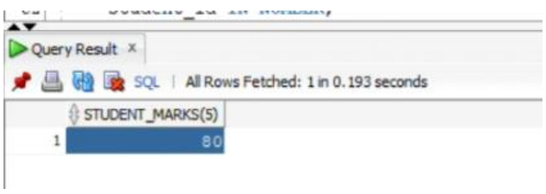
```
CREATE OR REPLACE FUNCTION student_marks(
    p_student_id NUMBER)
  RETURN NUMBER
IS
  v_total_marks NUMBER;
BEGIN
  SELECT total_marks
  INTO v_total_marks
  FROM students1
  WHERE student_id =p_student_id ;
  RETURN v_total_marks;
EXCEPTION
WHEN no_data_found THEN
  v_total_marks :=1000;
  return v_total_marks;
END;


select student_marks(2) from dual;
```



Query Result

All Rows Fetched: 8 in 0.191 seconds

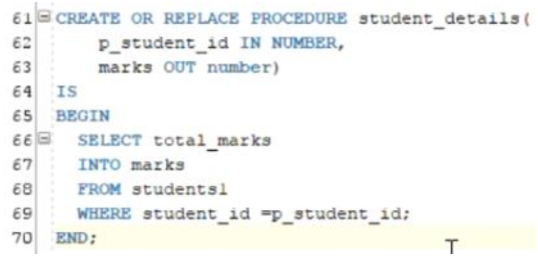
	STUDENT_ID	STUDENT_NAME	TOTAL_MARKS	SCHOOL_ID	GENDER
1	1	Student1	100	1	M
2	2	Student2	100	1	F
3	3	Student3	69	2	M
4	4	Student4	96	2	F
5	5	Student5	80	3	M
6	6	Student6	90	3	F
7	7	Student7	100	4	M
8	8	Student8	90	4	F



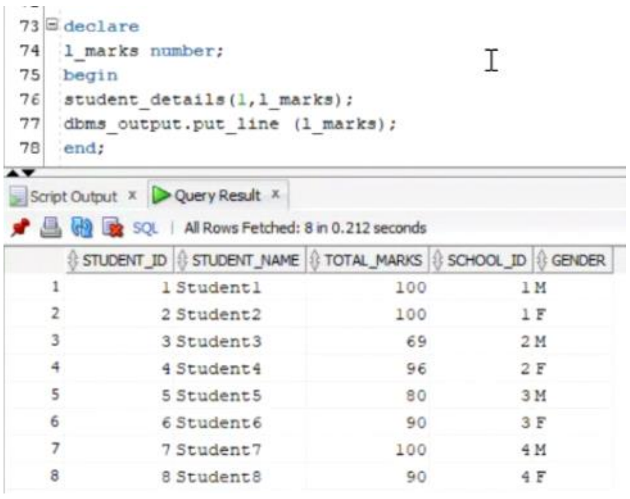
Query Result

All Rows Fetched: 1 in 0.193 seconds

	STUDENT_MARKS(5)
1	80



```
CREATE OR REPLACE PROCEDURE student_details(
    p_student_id IN NUMBER,
    marks OUT number)
IS
BEGIN
  SELECT total_marks
  INTO marks
  FROM students1
  WHERE student_id =p_student_id;
END;
```

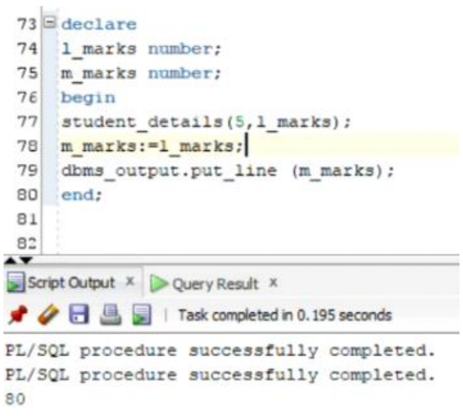


```
declare
l_marks number;
begin
student_details(1,l_marks);
dbms_output.put_line (l_marks);
end;
```

Script Output × Query Result ×

All Rows Fetched: 8 in 0.212 seconds

	STUDENT_ID	STUDENT_NAME	TOTAL_MARKS	SCHOOL_ID	GENDER
1	1	Student1	100	1	M
2	2	Student2	100	1	F
3	3	Student3	69	2	M
4	4	Student4	96	2	F
5	5	Student5	80	3	M
6	6	Student6	90	3	F
7	7	Student7	100	4	M
8	8	Student8	90	4	F

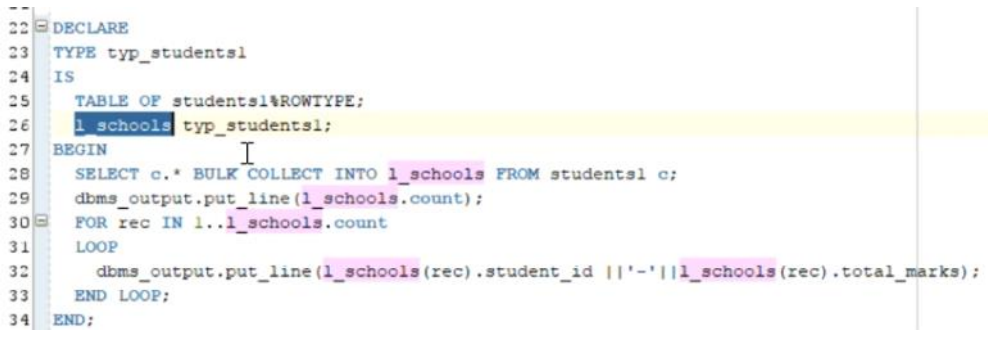


```
declare
l_marks number;
m_marks number;
begin
student_details(5,l_marks);
m_marks:=l_marks;
dbms_output.put_line (m_marks);
end;
```

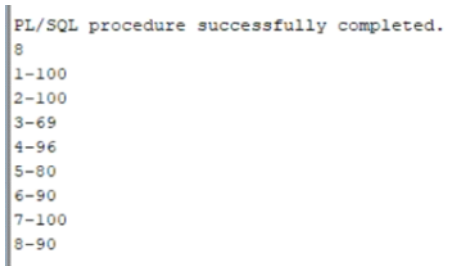
Script Output × Query Result ×

Task completed in 0.195 seconds

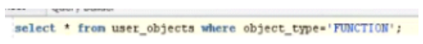
```
PL/SQL procedure successfully completed.
PL/SQL procedure successfully completed.
80
```



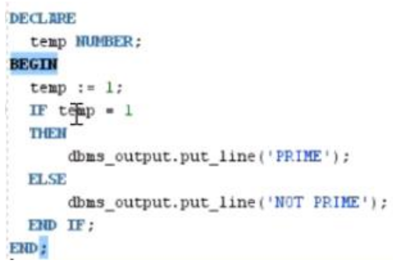
```
DECLARE
TYPE typ_students1
IS
  TABLE OF students1%ROWTYPE;
  l_schools typ_students1;
BEGIN
  SELECT c.* BULK COLLECT INTO l_schools FROM students1 c;
  dbms_output.put_line(l_schools.count);
  FOR rec IN 1..l_schools.count
  LOOP
    dbms_output.put_line(l_schools(rec).student_id ||'-'||l_schools(rec).total_marks);
  END LOOP;
END;
```



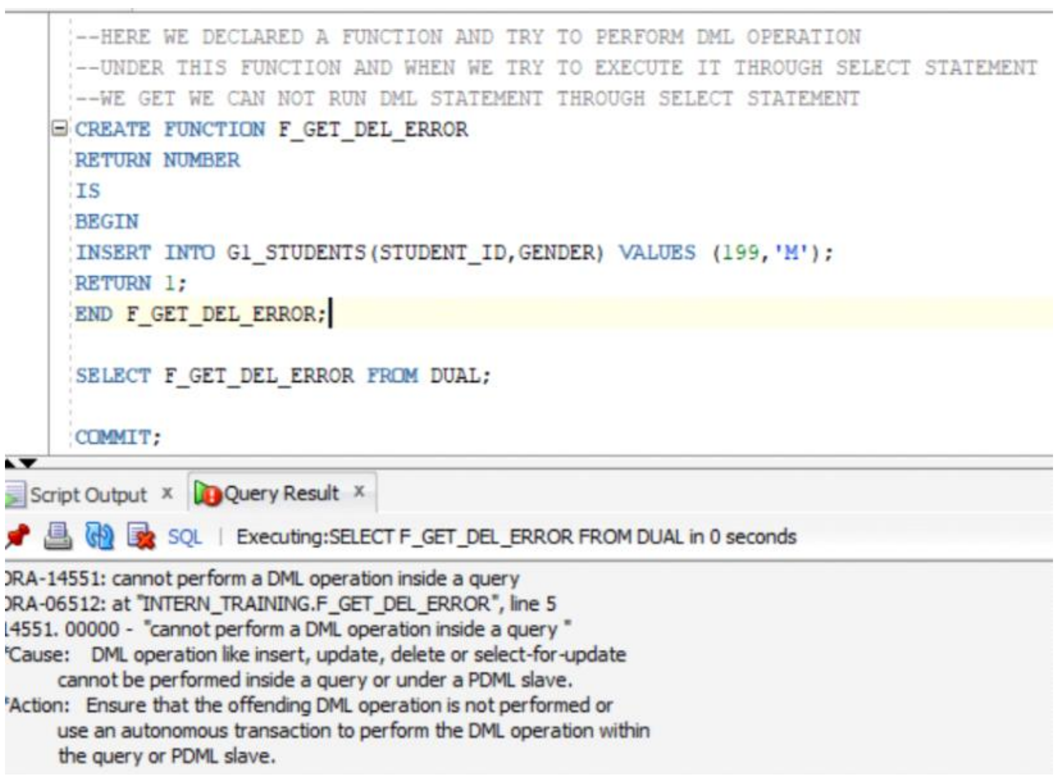
```
PL/SQL procedure successfully completed.
8
1-100
2-100
3-69
4-96
5-80
6-90
7-100
8-90
```



```
select * from user_objects where object_type='FUNCTION';
```



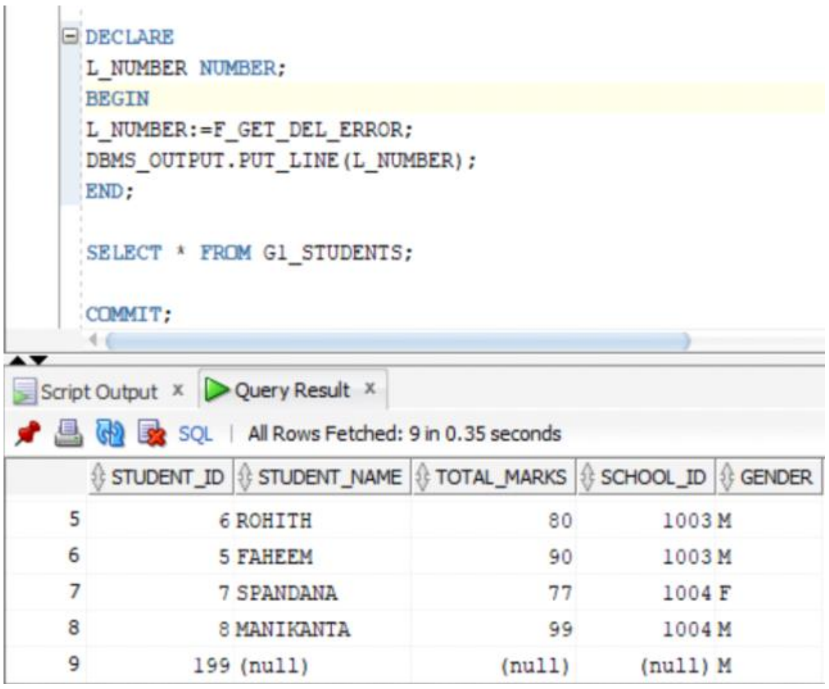
```
DECLARE
  temp NUMBER;
BEGIN
  temp := 1;
  IF temp = 1
  THEN
      dbms_output.put_line('PRIME');
  ELSE
      dbms_output.put_line('NOT PRIME');
  END IF;
END;
```



```
--HERE WE DECLARED A FUNCTION AND TRY TO PERFORM DML OPERATION
--UNDER THIS FUNCTION AND WHEN WE TRY TO EXECUTE IT THROUGH SELECT STATEMENT
--WE GET WE CAN NOT RUN DML STATEMENT THROUGH SELECT STATEMENT
CREATE FUNCTION F_GET_DEL_ERROR
RETURN NUMBER
IS
BEGIN
INSERT INTO G1_STUDENTS(STUDENT_ID,GENDER) VALUES (199,'M');
RETURN 1;
END F_GET_DEL_ERROR;

SELECT F_GET_DEL_ERROR FROM DUAL;

COMMIT;
```

```
Executing:SELECT F_GET_DEL_ERROR FROM DUAL in 0 seconds
RA-14551: cannot perform a DML operation inside a query
RA-06512: at "INTERN_TRAINING.F_GET_DEL_ERROR", line 5
4551. 00000 -  "cannot perform a DML operation inside a query "
Cause:    DML operation like insert, update, delete or select-for-update
          cannot be performed inside a query or under a PDML slave.
Action:   Ensure that the offending DML operation is not performed or
          use an autonomous transaction to perform the DML operation within
          the query or PDML slave.
```

But if we want to really use the function into our program then do the following



```
DECLARE
L_NUMBER NUMBER;
BEGIN
L_NUMBER:=F_GET_DEL_ERROR;
DBMS_OUTPUT.PUT_LINE(L_NUMBER);
END;

SELECT * FROM G1_STUDENTS;

COMMIT;
```

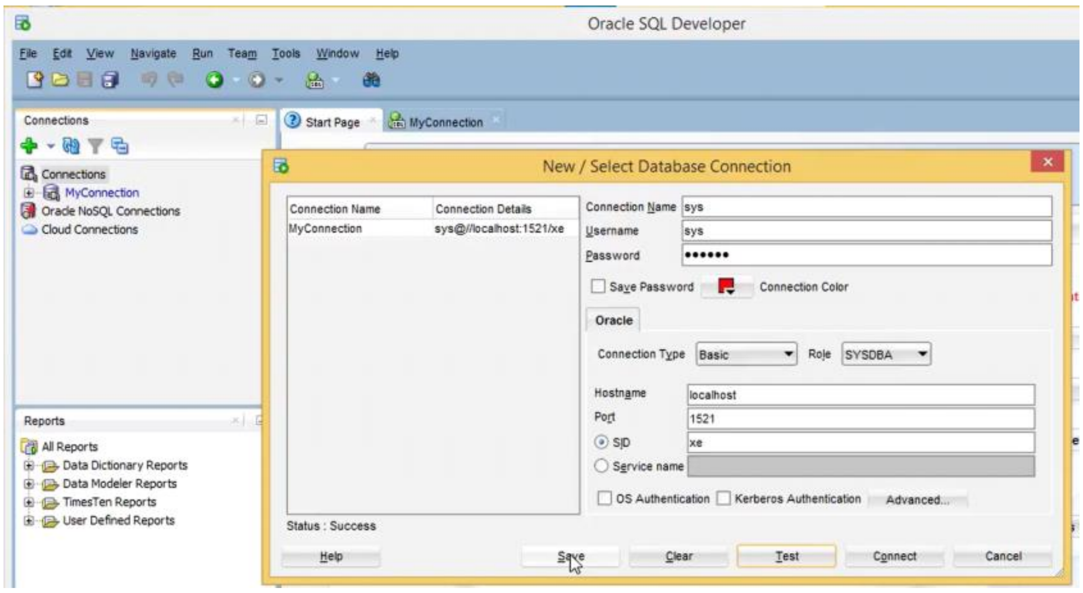
All Rows Fetched: 9 in 0.35 seconds

	STUDENT_ID	STUDENT_NAME	TOTAL_MARKS	SCHOOL_ID	GENDER
5	6	ROHITH	80	1003	M
6	5	FAHEEM	90	1003	M
7	7	SPANDANA	77	1004	F
8	8	MANIKANTA	99	1004	M
9	199	(null)	(null)	(null)	M

Self Learning DBMS

Tuesday, March 15, 2022 3:08 PM

How to create a connection in oracle?



New / Select Database Connection

Connection Name	Connection Details
MyConnection	sys@//localhost:1521/xe

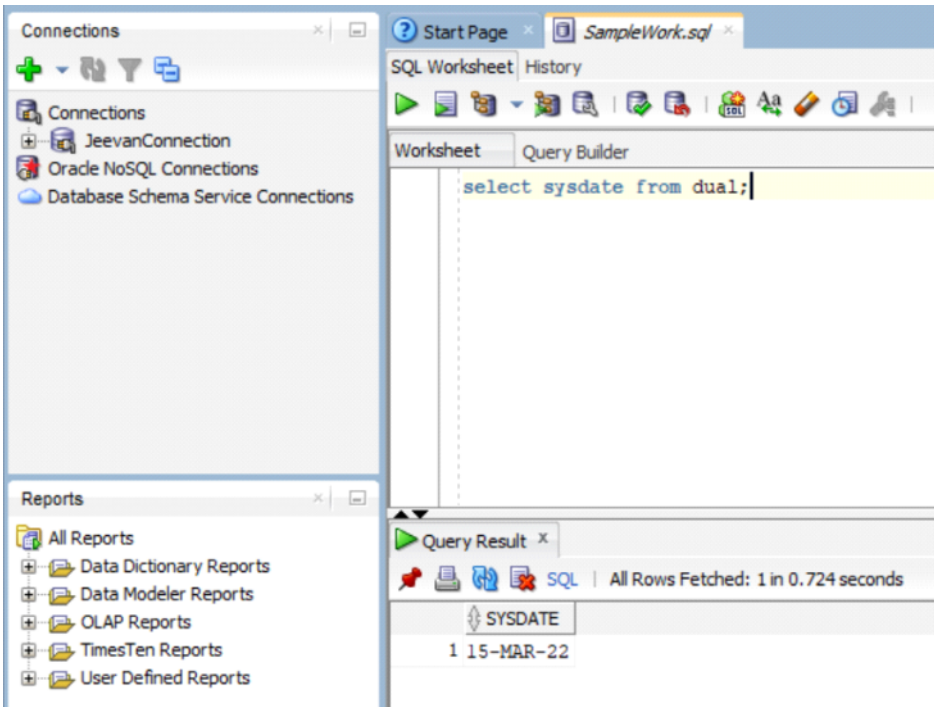
Connection Name: sys
Username: sys
Password: ••••••
Save Password [ ] Connection Color
Oracle
Connection Type: Basic Role: SYSDBA
Hostname: localhost
Port: 1521
(● SID) xe
( ) Service name
[ ] OS Authentication [ ] Kerberos Authentication Advanced...
Status : Success

- Here we can see on the top left corner a connection + symbol is present that is used to create a new connection and after we click it we get the following dialog box to fill the details.

How to disable un needed features?

- Tools>Features>And expand what you can see and disable what you do not need.

How to execute commands in Oracle

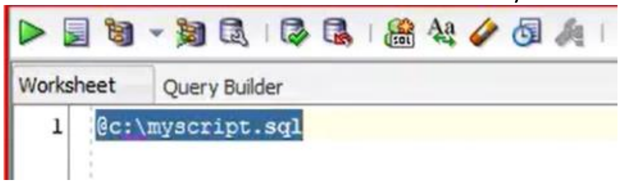


```
select sysdate from dual;
```

All Rows Fetched: 1 in 0.724 seconds

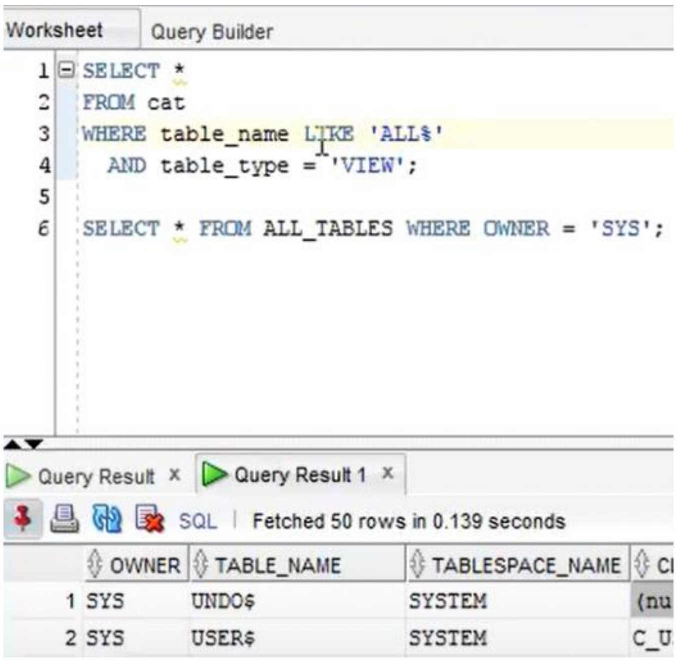
```
  SYSDATE
1 15-MAR-22
```

- First we need to write our command on the SQL file and then we need to click the play button to execute that.
- We can use CTRL+ENTER to run or F5 key



```
@c:\myscript.sql
```

- Or we can describe the sql file path and run that.
- For formatting any sql line we need to select sql line and then click ctrl+f7
- We can edit the preferences of formatting by Tools>preference > sql formatting>oracle formatting



```
SELECT *
FROM cat
WHERE table_name LIKE 'ALL$'
  AND table_type = 'VIEW';

SELECT * FROM ALL_TABLES WHERE OWNER = 'SYS';
```

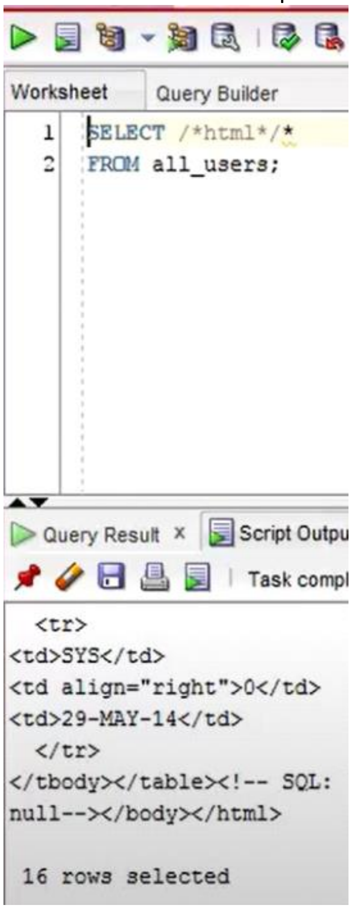
```
Fetched 50 rows in 0.139 seconds
```

	OWNER	TABLE_NAME	TABLESPACE_NAME	CI
1	SYS	UNDO$	SYSTEM	(nu
2	SYS	USER$	SYSTEM	C_U

- Here when we run different SQL commands we repeatedly get the result in the query result output tab. Hence we can pin that generated output by the symbol that we can see as red pin and execute other command then both the outputs will be present. Otherwise we can rename the output result then automatically for the new result we will get a different tab for the output.

How to export the data?

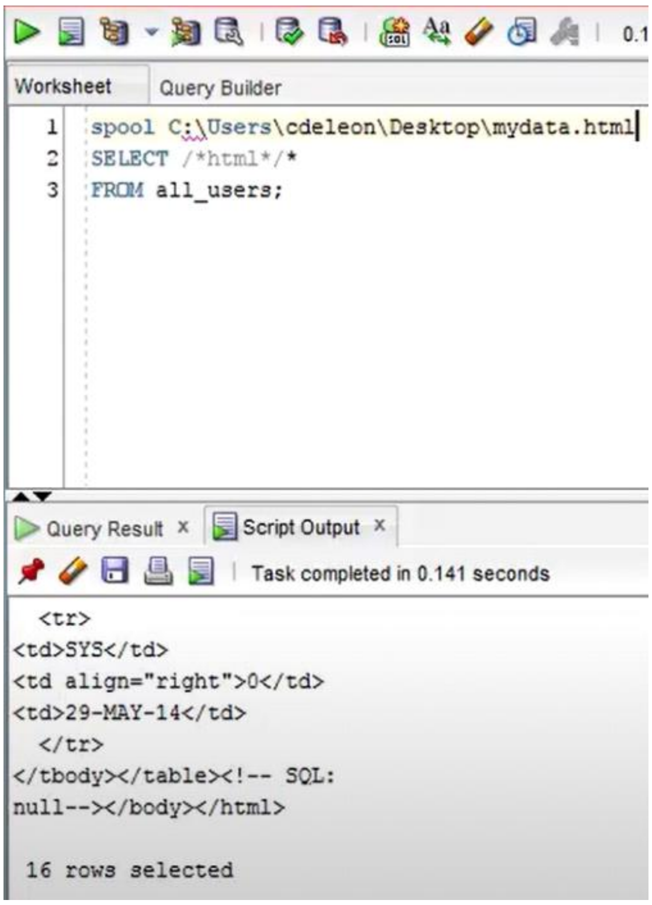
- Method 1: If we want to copy paste the output use Ctrl + Shift + C
- Method 2: If we want to export the data in any other format like excel word pdf or any format right click on the output and use the export option to export
- Method 3: Use simple annotation to get the desired output as following.



```
SELECT /*html*/*
FROM all_users;
```

```
  <tr>
<td>SYS</td>
<td align="right">0</td>
<td>29-MAY-14</td>
  </tr>
</tbody></table><!-- SQL:
null--></body></html>

 16 rows selected
```

- Method 4: If we want to export the data using annotation and have a file of it use spooling technique

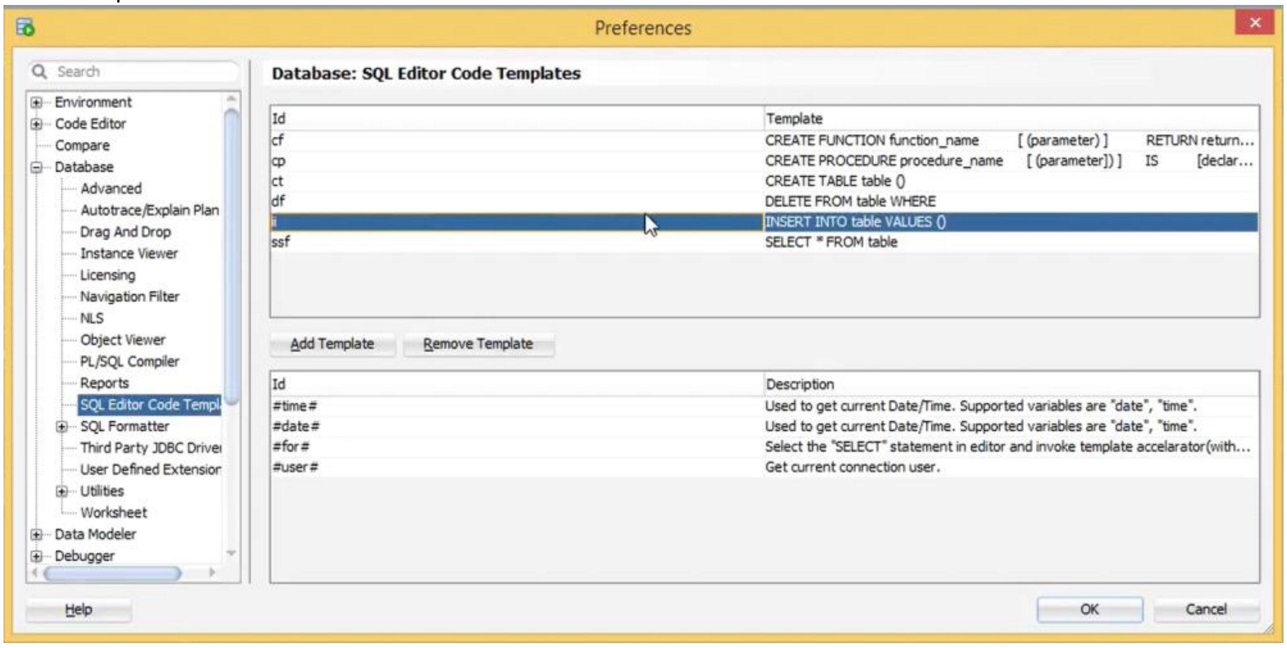


```
spool C:\Users\cdeleon\Desktop\mydata.html
SELECT /*html*/*
FROM all_users;
```

Task completed in 0.141 seconds

```
  <tr>
<td>SYS</td>
<td align="right">0</td>
<td>29-MAY-14</td>
  </tr>
</tbody></table><!-- SQL:
null--></body></html>

 16 rows selected
```

Code Templates:

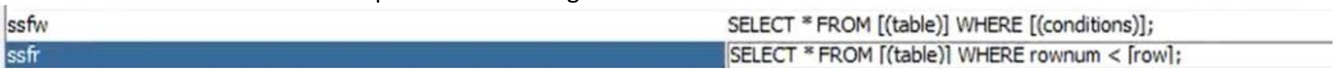


Database: SQL Editor Code Templates

Id	Template
cf	CREATE FUNCTION function_name [(parameter)] RETURN return…
cp	CREATE PROCEDURE procedure_name [(parameter])] IS [declar…
ct	CREATE TABLE table ()
df	DELETE FROM table WHERE
ii	INSERT INTO table VALUES ()
ssf	SELECT * FROM table

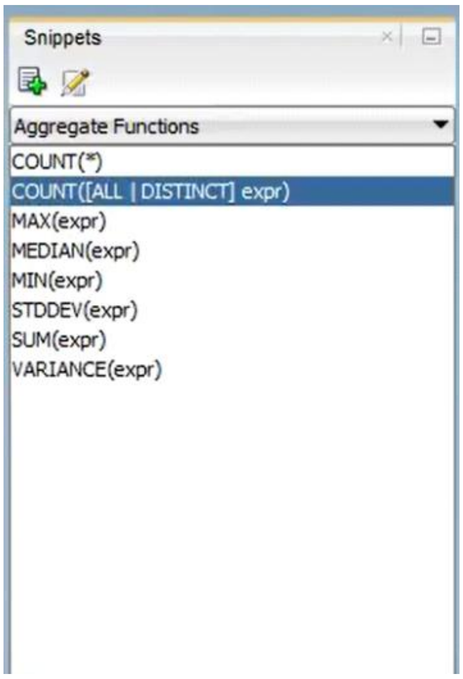
Id	Description
#time#	Used to get current Date/Time. Supported variables are "date", "time".
#date#	Used to get current Date/Time. Supported variables are "date", "time".
#for#	Select the "SELECT" statement in editor and invoke template accelarator(with…
#user#	Get current connection user.

- For example, if we type ii and click CTRL+SPACE then automatically according to the code template it changes to insert statement. Similarly we are having different statements and their templates defined as above.
- We can create our own templates as following



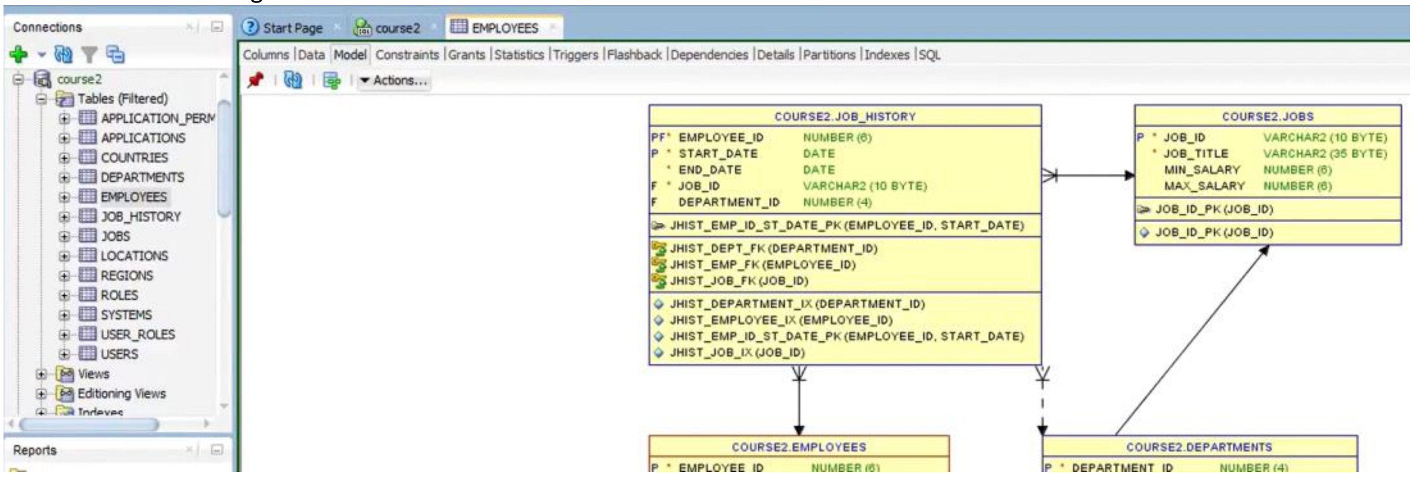
ssfw	SELECT * FROM [(table)] WHERE [(conditions)];
ssfr	SELECT * FROM [(table)] WHERE rownum < [row];

- The benefit of it is we no need to define the whole and after getting the statement using tab space we can shift from parameters that we want to enter one by one.
- If we want more snippets we can use pre defined snippets View>Snippets



COUNT(*)
COUNT([ALL | DISTINCT] expr)
MAX(expr)
MEDIAN(expr)
MIN(expr)
STDDEV(expr)
SUM(expr)
VARIANCE(expr)

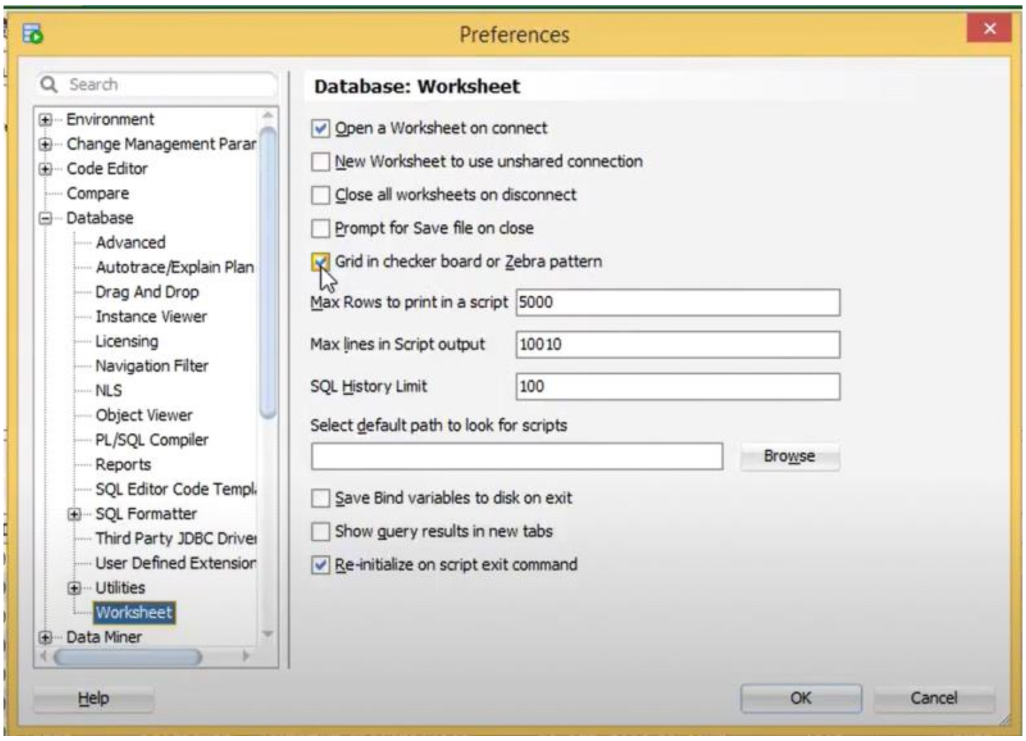
Here if we want to get model diagrams then we must go the connections column click the table that we want to check and go to the model tab of that table as shown below



```
COURSE2.JOB_HISTORY
PF * EMPLOYEE_ID     NUMBER (6)
P  * START_DATE      DATE
   * END_DATE        DATE
F  * JOB_ID          VARCHAR2 (10 BYTE)
F    DEPARTMENT_ID   NUMBER (4)
JHIST_EMP_ID_ST_DATE_PK (EMPLOYEE_ID, START_DATE)
JHIST_DEPT_FK (DEPARTMENT_ID)
JHIST_EMP_FK (EMPLOYEE_ID)
JHIST_JOB_FK (JOB_ID)
JHIST_DEPARTMENT_IX (DEPARTMENT_ID)
JHIST_EMPLOYEE_IX (EMPLOYEE_ID)
JHIST_EMP_ID_ST_DATE_PK (EMPLOYEE_ID, START_DATE)
JHIST_JOB_IX (JOB_ID)

COURSE2.JOBS
P * JOB_ID        VARCHAR2 (10 BYTE)
  * JOB_TITLE     VARCHAR2 (35 BYTE)
    MIN_SALARY    NUMBER (6)
    MAX_SALARY    NUMBER (6)
JOB_ID_PK (JOB_ID)
JOB_ID_PK (JOB_ID)

COURSE2.EMPLOYEES
P * EMPLOYEE_ID   NUMBER (6)

COURSE2.DEPARTMENTS
P * DEPARTMENT_ID   NUMBER (4)
```

Here if we want to check the relational model diagram of only some of the tables then we must do go to browser tab on the screen and click the relational table and right click that and create new relational table screen and now drag all the tables that we want to check of the relations . If we drag the tables on the screen we will get relations that they are into automatically.

- If we want to print more rows in the output script file then we should change the preference under Tools>Preference>Worksheet



Database: Worksheet

- [✓] Open a Worksheet on connect
- [ ] New Worksheet to use unshared connection
- [ ] Close all worksheets on disconnect
- [ ] Prompt for Save file on close
- [✓] Grid in checker board or Zebra pattern

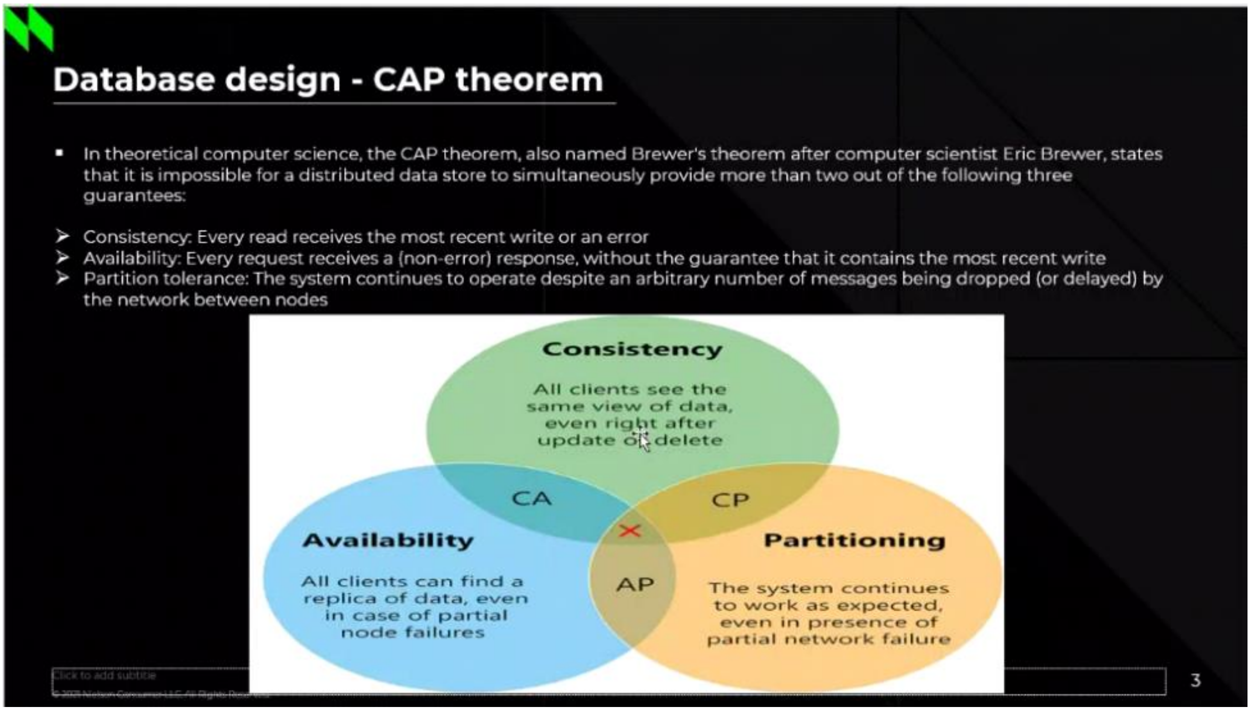
Max Rows to print in a script	5000
Max lines in Script output	10010
SQL History Limit	100
Select default path to look for scripts	

- [ ] Save Bind variables to disk on exit
- [ ] Show query results in new tabs
- [✓] Re-initialize on script exit command

When we type the query if we give CTRL+SPACE then we would get a drop down of * suggestions.
CTRL+G is used to go to the line
Shift+F4 is used to describe the table similar to describe table_name
Shift + Alt +F5 move right tab
Shift + F5 move left tab
CTRL + Tab move to last worksheet.
CTRL + SHIFT + ' is used to switch the case of letters
CTRL + / used for commenting

NoSQL Class Notes

Thursday, March 17, 2022 10:37 AM



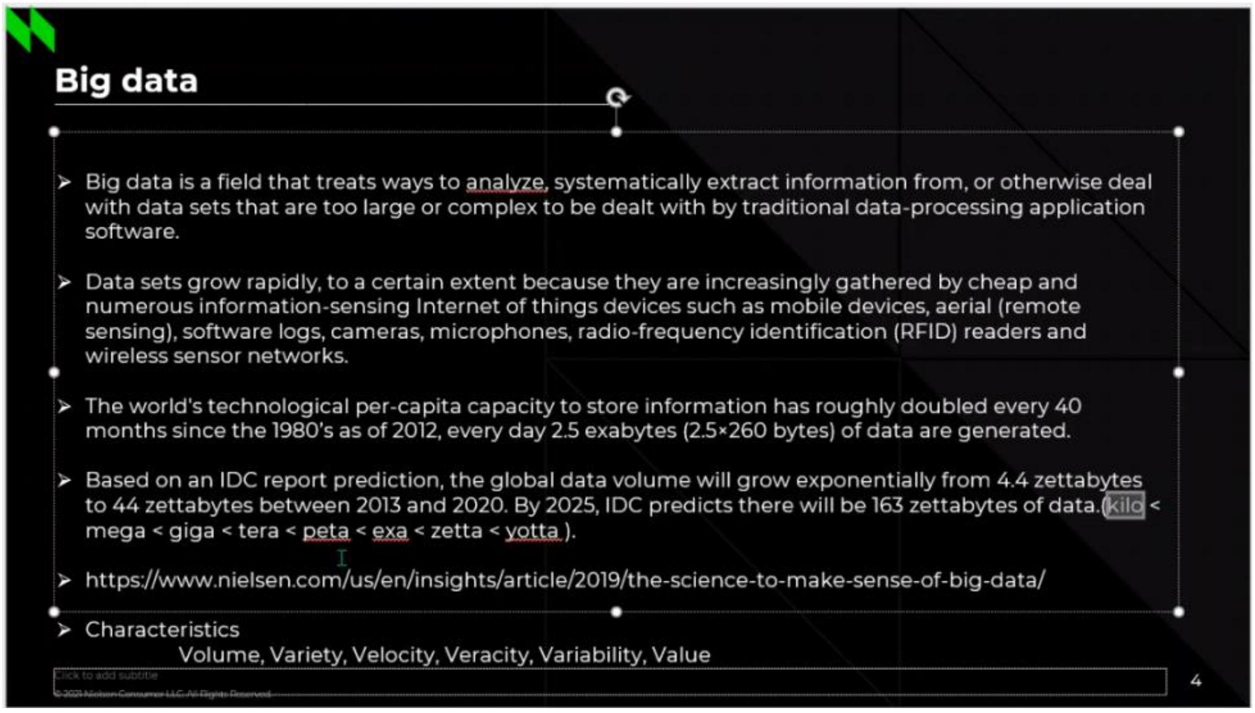
Database design - CAP theorem

- In theoretical computer science, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

- Consistency: Every read receives the most recent write or an error
- Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Consistency
All clients see the same view of data, even right after update or delete
CA
CP
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure

3

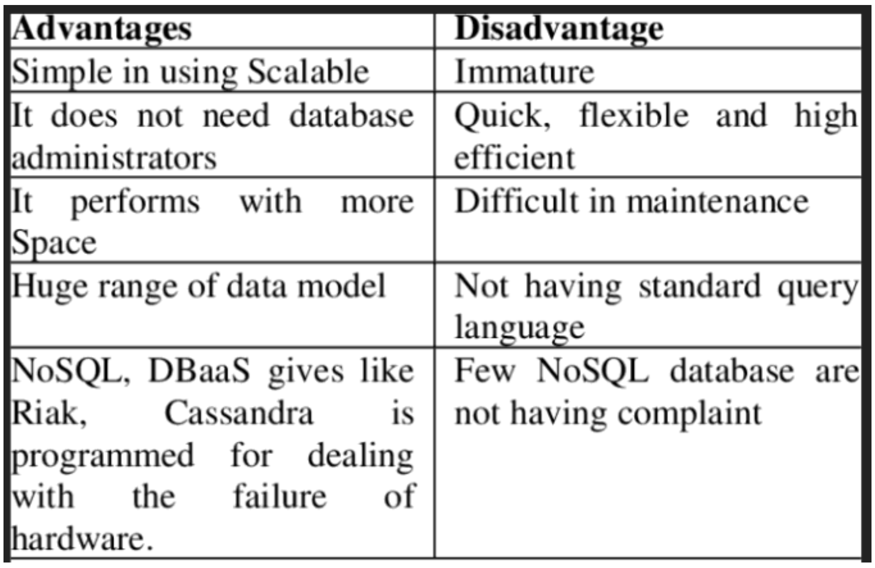


Big data

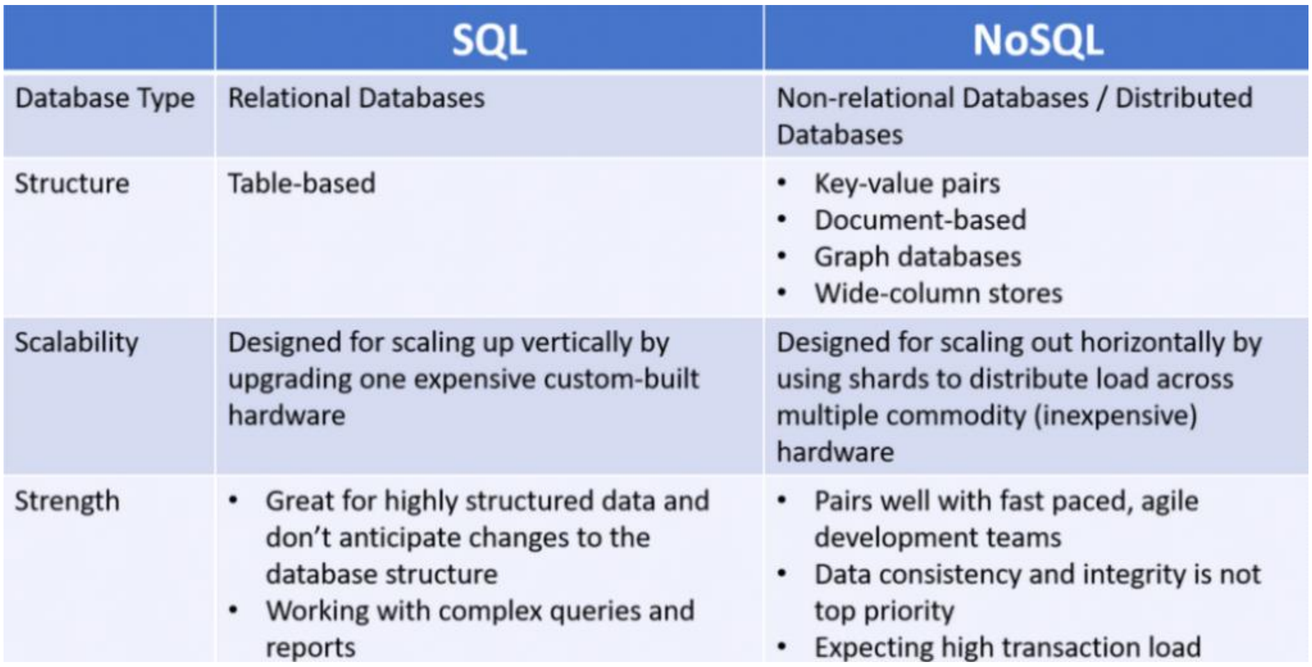
- Big data is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software.
- Data sets grow rapidly, to a certain extent because they are increasingly gathered by cheap and numerous information-sensing Internet of things devices such as mobile devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks.
- The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980's as of 2012, every day 2.5 exabytes (2.5×260 bytes) of data are generated.
- Based on an IDC report prediction, the global data volume will grow exponentially from 4.4 zettabytes to 44 zettabytes between 2013 and 2020. By 2025, IDC predicts there will be 163 zettabytes of data.(kilo < mega < giga < tera < peta < exa < zetta < yotta).
- https://www.nielsen.com/us/en/insights/article/2019/the-science-to-make-sense-of-big-data/
- Characteristics
  Volume, Variety, Velocity, Veracity, Variability, Value

4

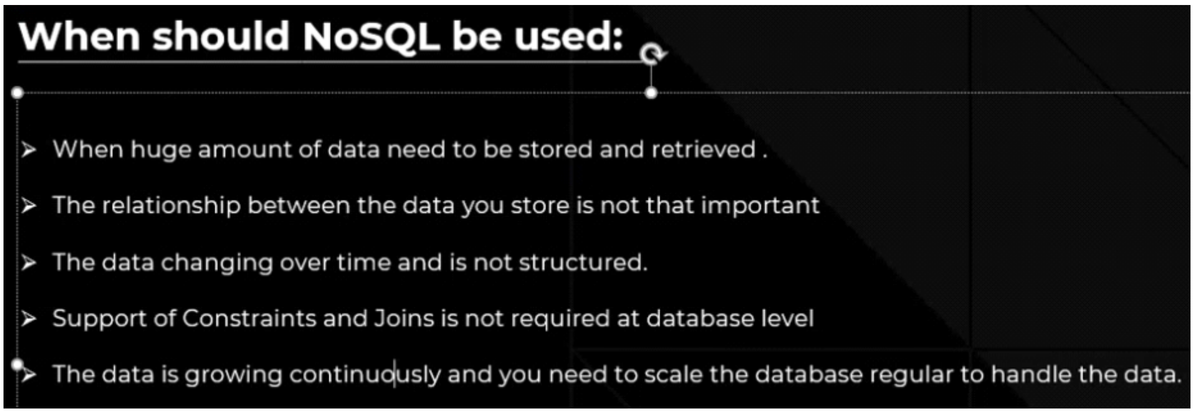
If we want to establish relations among the data and tables Relational databases are required where as coming to only for storage purpose no SQL database language is enough.



Advantages	Disadvantage
Simple in using Scalable	Immature
It does not need database administrators	Quick, flexible and high efficient
It performs with more Space	Difficult in maintenance
Huge range of data model	Not having standard query language
NoSQL, DBaaS gives like Riak, Cassandra is programmed for dealing with the failure of hardware.	Few NoSQL database are not having complaint



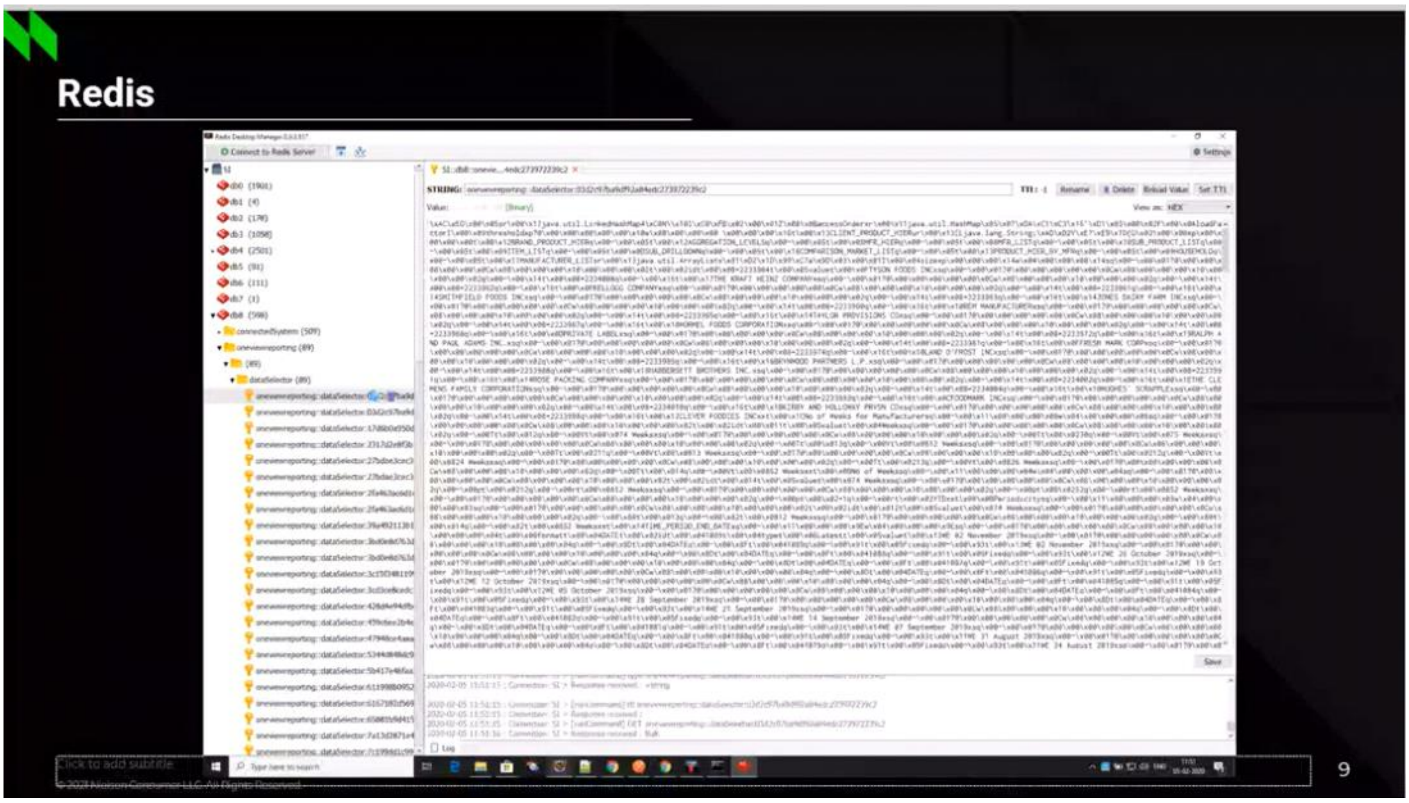
	SQL	NoSQL
Database Type	Relational Databases	Non-relational Databases / Distributed Databases
Structure	Table-based	• Key-value pairs • Document-based • Graph databases • Wide-column stores
Scalability	Designed for scaling up vertically by upgrading one expensive custom-built hardware	Designed for scaling out horizontally by using shards to distribute load across multiple commodity (inexpensive) hardware
Strength	• Great for highly structured data and don't anticipate changes to the database structure • Working with complex queries and reports	• Pairs well with fast paced, agile development teams • Data consistency and integrity is not top priority • Expecting high transaction load



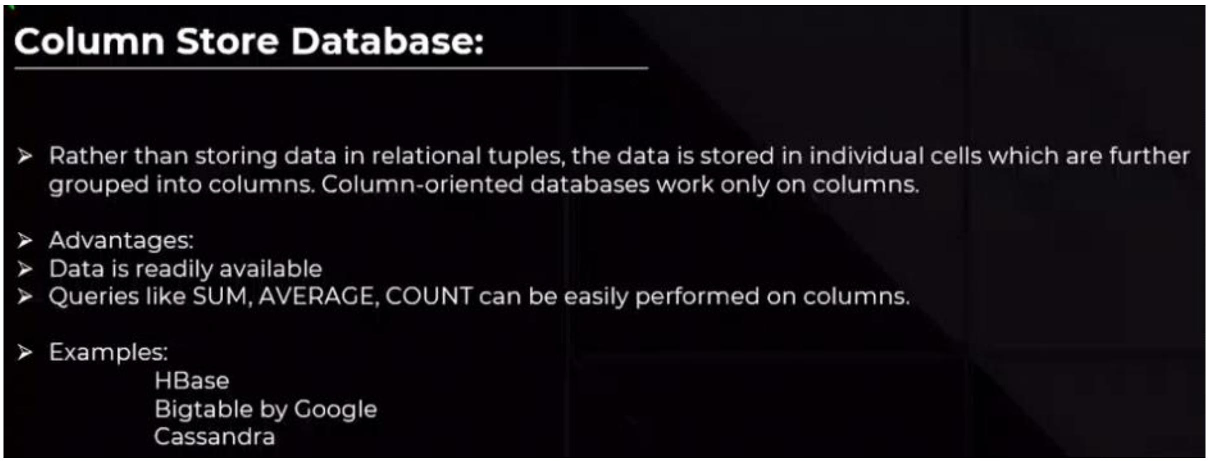
When should NoSQL be used:

- When huge amount of data need to be stored and retrieved .
- The relationship between the data you store is not that important
- The data changing over time and is not structured.
- Support of Constraints and Joins is not required at database level
- The data is growing continuously and you need to scale the database regular to handle the data.

Key value database - Redis
Document based database - Mango DB
Column based database - Cassandra
Graph database - neoforgue

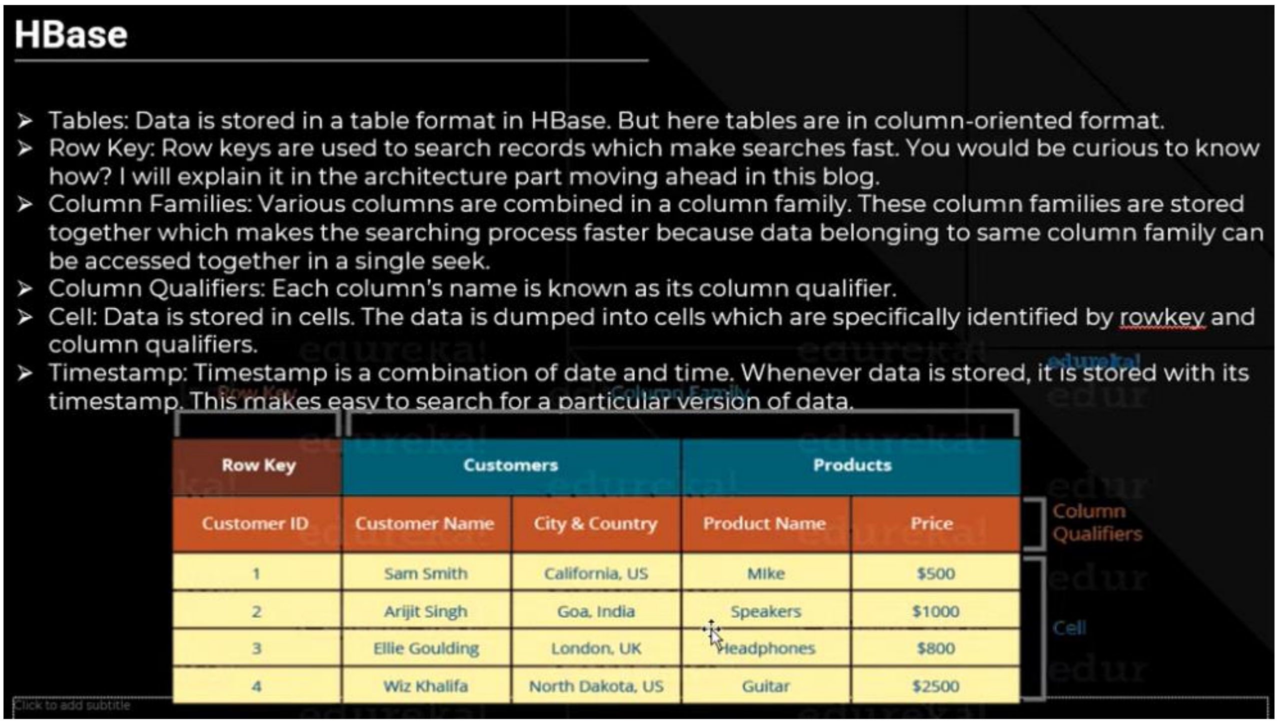


Redis



Column Store Database:

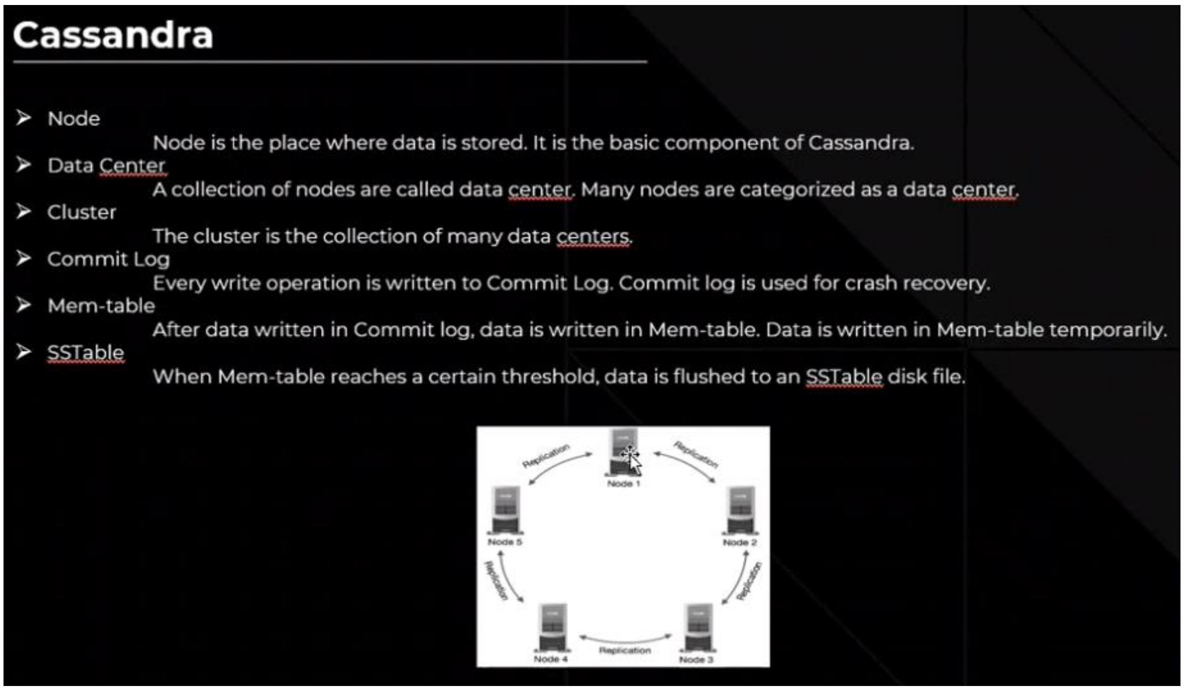
- ➢ Rather than storing data in relational tuples, the data is stored in individual cells which are further grouped into columns. Column-oriented databases work only on columns.
- ➢ Advantages:
- ➢ Data is readily available
- ➢ Queries like SUM, AVERAGE, COUNT can be easily performed on columns.
- ➢ Examples:
  - HBase
  - Bigtable by Google
  - Cassandra



HBase

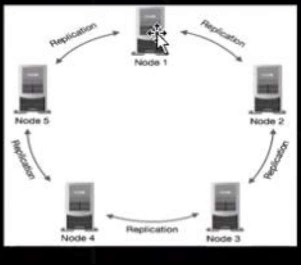
- ➢ Tables: Data is stored in a table format in HBase. But here tables are in column-oriented format.
- ➢ Row Key: Row keys are used to search records which make searches fast. You would be curious to know how? I will explain it in the architecture part moving ahead in this blog.
- ➢ Column Families: Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.
- ➢ Column Qualifiers: Each column's name is known as its column qualifier.
- ➢ Cell: Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.
- ➢ Timestamp: Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

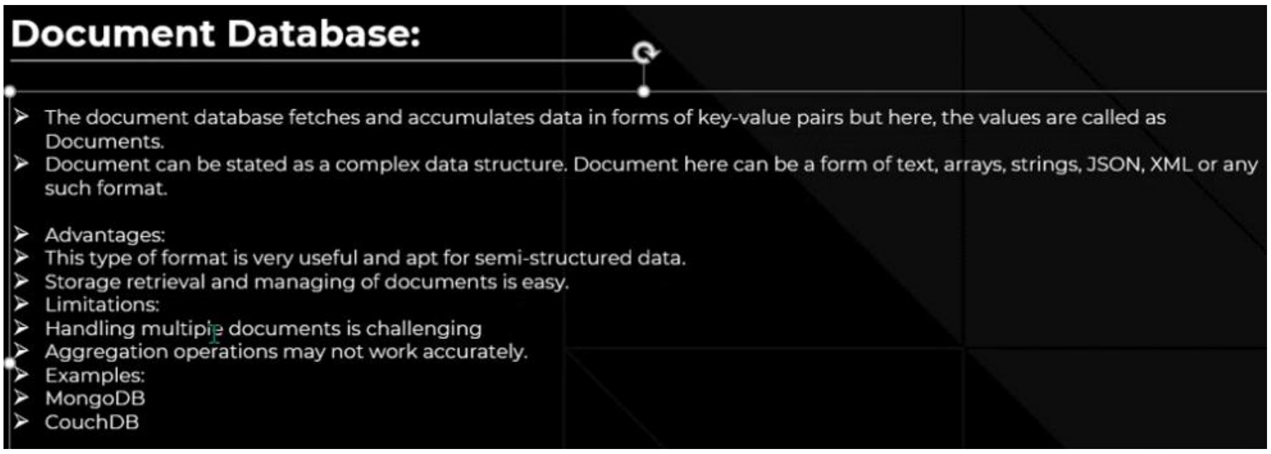
Row Key	Customers		Products	
Customer ID	Customer Name	City & Country	Product Name	Price
1	Sam Smith	California, US	Mike	$500
2	Arijit Singh	Goa, India	Speakers	$1000
3	Ellie Goulding	London, UK	Headphones	$800
4	Wiz Khalifa	North Dakota, US	Guitar	$2500



Cassandra

- Node
  Node is the place where data is stored. It is the basic component of Cassandra.
- Data Center
  A collection of nodes are called data center. Many nodes are categorized as a data center.
- Cluster
  The cluster is the collection of many data centers.
- Commit Log
  Every write operation is written to Commit Log. Commit log is used for crash recovery.
- Mem-table
  After data written in Commit log, data is written in Mem-table. Data is written in Mem-table temporarily.
- SSTable
  When Mem-table reaches a certain threshold, data is flushed to an SSTable disk file.



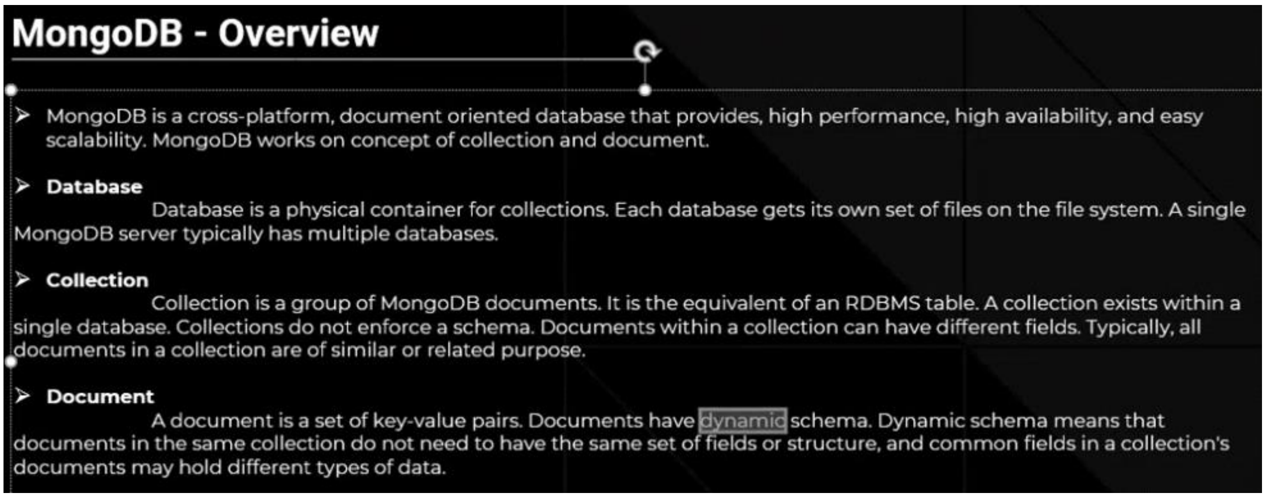


Document Database:

- The document database fetches and accumulates data in forms of key-value pairs but here, the values are called as Documents.
- Document can be stated as a complex data structure. Document here can be a form of text, arrays, strings, JSON, XML or any such format.

- Advantages:
- This type of format is very useful and apt for semi-structured data.
- Storage retrieval and managing of documents is easy.
- Limitations:
- Handling multiple documents is challenging
- Aggregation operations may not work accurately.
- Examples:
- MongoDB
- CouchDB

- Mango do support semi structured data but relational databases only support strictly structured data.



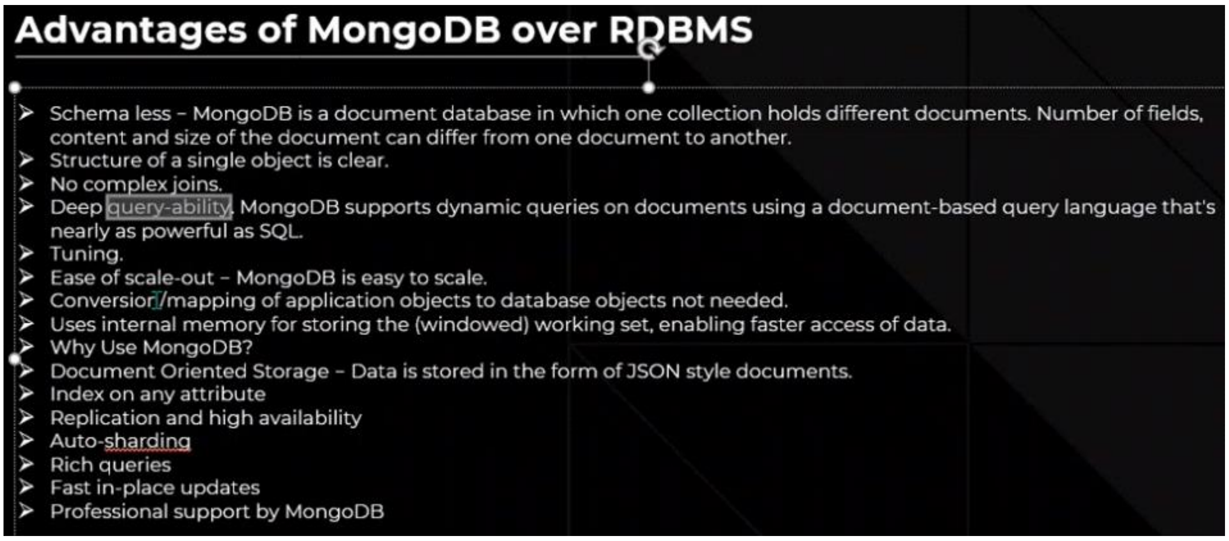
MongoDB - Overview

- MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

- **Database**
  Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

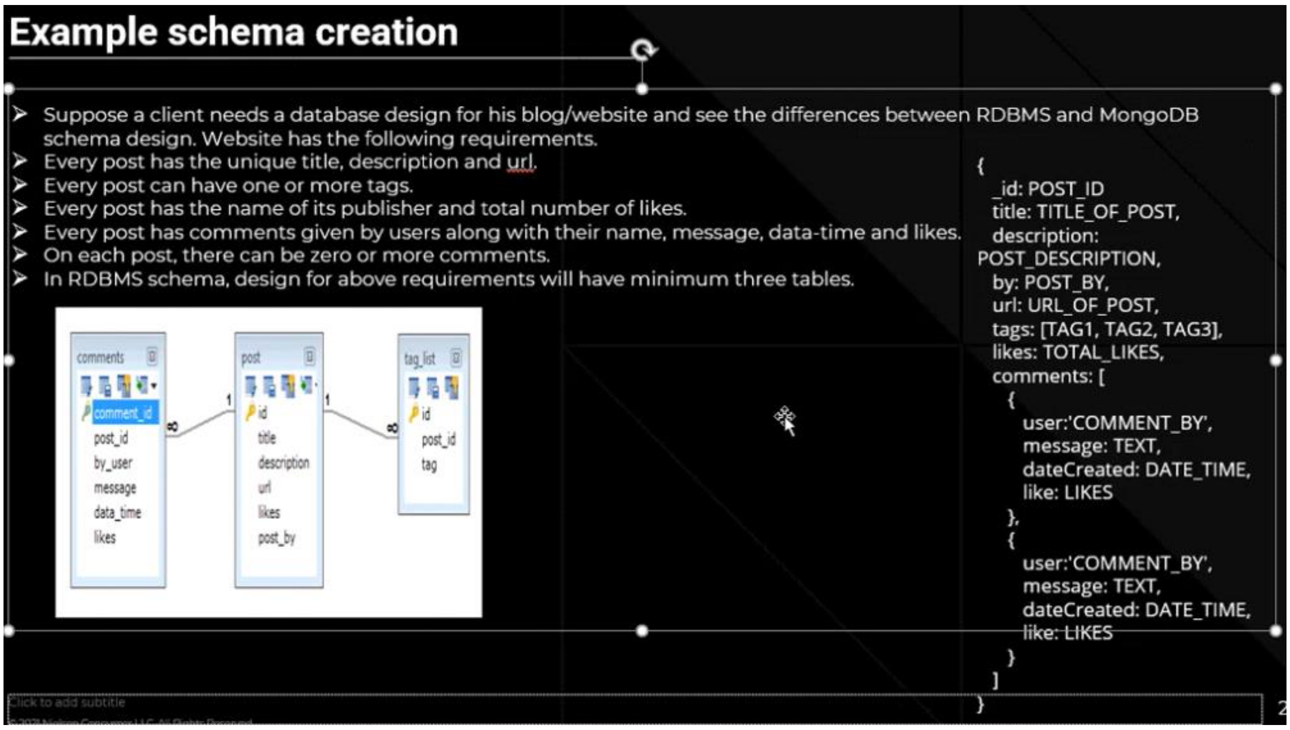
- **Collection**
  Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

- **Document**
  A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.



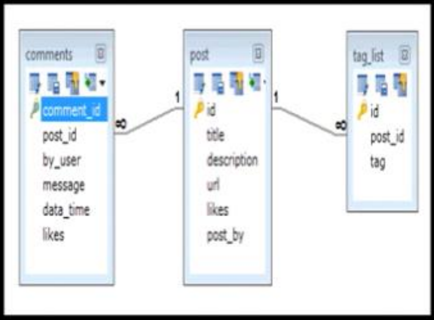
Advantages of MongoDB over RDBMS

- ➢ Schema less – MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- ➢ Structure of a single object is clear.
- ➢ No complex joins.
- ➢ Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- ➢ Tuning.
- ➢ Ease of scale-out – MongoDB is easy to scale.
- ➢ Conversion/mapping of application objects to database objects not needed.
- ➢ Uses internal memory for storing the (windowed) working set, enabling faster access of data.
- ➢ Why Use MongoDB?
- ➢ Document Oriented Storage – Data is stored in the form of JSON style documents.
- ➢ Index on any attribute
- ➢ Replication and high availability
- ➢ Auto-sharding
- ➢ Rich queries
- ➢ Fast in-place updates
- ➢ Professional support by MongoDB



Example schema creation

- ➢ Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.
- ➢ Every post has the unique title, description and url.
- ➢ Every post can have one or more tags.
- ➢ Every post has the name of its publisher and total number of likes.
- ➢ Every post has comments given by users along with their name, message, data-time and likes.
- ➢ On each post, there can be zero or more comments.
- ➢ In RDBMS schema, design for above requirements will have minimum three tables.



```
{
  _id: POST_ID
  title: TITLE_OF_POST,
  description:
POST_DESCRIPTION,
  by: POST_BY,
  url: URL_OF_POST,
  tags: [TAG1, TAG2, TAG3],
  likes: TOTAL_LIKES,
  comments: [
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    },
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    }
  ]
}
```

Click to add subtitle