

DDL (Data Definition Language)

- Create
- Alter
- Drop
- Truncate

• DML (Data Manipulation Language)

- Insert
- Update
- Delete
- Select

• DCL (Data Control Language)

- Grant
- Revoke

• TCL (Transaction Control Language)

- Commit
- Rollback
- Savepoint

Integers

Decimal numbers--- NUMBER, INTEGER .

Number is an oracle data type. Integer is an ANSI data type. Integer is equivalent of NUMBER(38)
The syntax for NUMBER is NUMBER(P,S) p is the precision and s is the scale. P can range from 1 to 38 and s from -84 to 127

Floating point numbers---- FLOAT

Fixed length character strings---- CHAR (len)

Fixed length character data of length len bytes. This should be used for fixed length data.

Variable length character strings --- Varchar2(len)

Variable length character string having maximum length *len* bytes. We must specify the size

Dates-----DATE

NULL

- Missing/unknown/inapplicable data represented as a **null** value
- NULL is not a data value. It is just an indicator that the value is unknown

SQL - CREATE TABLE

Syntax:

```
CREATE TABLE tablename  
(  
column_name data_type constraints, ...  
)
```

```
CREATE TABLE Customer_Details(  
Cust_ID Number(5) CONSTRAINT Nnull1 NOT NULL,  
Cust_Last_Name VarChar2(20) CONSTRAINT Nnull2 NOT NULL,  
Cust_Mid_Name VarChar2(4),  
Cust_First_Name VarChar2(20),  
Account_No Number(5) CONSTRAINT Pkey1 PRIMARY KEY,  
Account_Type VarChar2(10) CONSTRAINT Nnull3 NOT NULL,  
Bank_Branch VarChar2(25) CONSTRAINT Nnull4 NOT NULL,  
Cust_Email VarChar2(30)  
);
```

```
CREATE TABLE Customer_Details(  
Cust_ID Number(5) CONSTRAINT Nnull7 NOT NULL,  
Cust_Last_Name VarChar2(20) CONSTRAINT Nnull8 NOT NULL,  
Cust_Mid_Name VarChar2(4),  
Cust_First_Name VarChar2(20),  
Account_No Number(5) CONSTRAINT Nnull9 NOT NULL,  
Account_Type VarChar2(10) CONSTRAINT Nnull10 NOT NULL,  
Bank_Branch VarChar2(25) CONSTRAINT Nnull11 NOT NULL,  
Cust_Email VarChar2(30),  
CONSTRAINT PKey3 PRIMARY KEY(Cust_ID,Account_No)  
);
```

```
CREATE TABLE EMPLOYEE_MANAGER(  
Employee_ID Number(6) CONSTRAINT Pkey2 PRIMARY KEY,  
Employee_Last_Name VarChar2(25),  
Employee_Mid_Name VarChar2(5),  
Employee_First_Name VarChar2(25),  
Employee_Email VarChar2(35),  
Department VarChar2(10),  
Grade Number(2),  
MANAGER_ID Number(6) CONSTRAINT Fkey2  
REFERENCES EMPLOYEE_MANAGER(Employee_ID)  
);
```

```
ALTER TABLE Customer_Details  
ADD Contact_Phone Char(10);
```

ALTER TABLE Customer_Details
MODIFY Contact_Phone **Char(12);**

ALTER TABLE Customer_Details

DROP (Contact_Phone);

INSERT INTO Customer_Details
VALUES (106, 'Costner', 'A.', 'Kevin', 3350, 'Savings', 'Indus Bank',
'Costner_Kevin@times.com')

Syntax: INSERT INTO tablename (Columnlist) VALUES (value list)

• Inserting one row, few columns at a time

INSERT INTO Customer_Details
(Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name, Account_No,
Account_Type, Bank_Branch)
VALUES (107, 'Robert', 'B.', 'Dan', 3351, 'Savings', 'Indus Bank')

Or

INSERT INTO Customer_Details
(Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name, Account_No,
Account_Type, Bank_Branch, **Cust_Email**)
VALUES (108, 'Robert', 'B.', 'Dan', 3352, 'Savings', 'Indus Bank', **NULL**)

Deleting All Rows

DELETE FROM Customer_Details

Deleting Specific Rows

DELETE
FROM Customer_Details
WHERE Cust_ID = 102

UPDATE Customer_Fixed_Deposit
SET Rate_of_Interest_in_Percent = **NULL**;

Updating Particular rows

UPDATE Customer_Fixed_Deposit
SET Rate_of_Interest_in_Percent = 7.3
WHERE Amount_in_Dollars > 3000;

Updating Multiple Columns

UPDATE Customer_Fixed_Deposit
SET Cust_Email = 'Quails_Jack@rediffmail.com',
Rate_of_Interest_in_Percent = 7.3
WHERE Cust_ID = 104

SELECT *
FROM Customer_Details;

Retrieving Few Columns

SELECT Cust_ID, Account_No
FROM Customer_Details;

Implementing Customized Columns Names

SELECT Account_No **AS** "Customer Account No.",
Total_Available_Balance_in_Dollars **AS** "Total Balance"
FROM Customer_Transaction;

TRUNCATE

TRUNCATE removes all rows from a table. The operation cannot be rolled back and no triggers will be fired. As such, TRUNCATE is faster and doesn't use as much undo space as a DELETE.

```
SQL> TRUNCATE TABLE emp;
```

Table truncated.

```
SQL> SELECT COUNT(*) FROM emp;
```

```
COUNT(*)
-----
0
```

1>TRUNCATE is a DDL command whereas DELETE is a DML command.

2>TRUNCATE is much faster than DELETE.

Reason:When you type DELETE.all the data get copied into the Rollback Tablespace first.then delete operation get performed.Thatswhy when you type ROLLBACK after deleting a table ,you can get back the data(The system get it for you from the Rollback Tablespace).All this process take time.But when you type TRUNCATE,it removes data directly without copying it into the Rollback Tablespace.Thatswhy TRUNCATE is faster.Once you Truncate you can't get back the data.

3>You can't rollback in TRUNCATE but in DELETE you can rollback.TRUNCATE removes the record permanently.

A constraint is a rule that restricts the values that may be present in the database. Constraints can be mainly classified in to two categories.

Not Null constraint

A null value indicates 'not applicable', 'missing', or 'not known'. A null value is distinct from zero or blank space.

A column, defined as a not null, cannot have a null value.

Such a column become a mandatory (compulsory) column and cannot be left empty for any record.

Syntax : ColumName datatype (size) NOT NULL

Example :

```
create table Account (ano char(3),
```

```
Balance number(9) NOT NULL, Branch varchar2(10));
```

Now, if we insert NULL value to Balance column then it will generate an error.

Check constraint

The check constraint is used to implement business rule. So, it is also called business rule constraint. Example of business rule: A balance in any account should not be negative.

Business rule define a domain for a particular column.

The check constraint is bound to a particular column.

Once check constraint is implemented, any insert or update operation on that table must follow this constraint.

?? If any operation violates condition, it will be rejected.

?? **Syntax :** ColumnName datatype (size) check(condition) ?? **Example :**

create table Account (ano char(3),

Balance number(9) CHECK (not (balance <0)), Branch varchar2(10));

?? Any business rule validations can be applied using this constraint. ?? A condition must be some valid logical expression.

?? A check constraint takes longer time to execute.

?? On violation of this constrain, oracle display error message like – “check constraint violated”.

Unique Constraint

Sometime there may be requirement that column cannot contain duplicate values. ?? A column, defined as a unique, cannot have duplicate values across all records.

Syntax : ColumnName datatype (size) UNIQUE ??

Example :

create table Account (ano char(3) UNIQUE, Balance number(9), Branch varchar2(10));

Though, a unique constraint does not allow duplicate values, NULL values can be duplicated in a column defined as a UNIQUE column.

A table can have more than one column defined as a unique column.

If multiple columns need to be defined as composite unique column, then only table level definition is applicable.

Maximum 16 columns can be combined as a composite unique key in a table.

Primary key Constraint

“A primary key is a set of one or more columns used to identify each record uniquely in a column.”

?? A single column primary key is called a simple key, while a multi-column primary key is called composite primary key.

?? Oracle provides a primary key constraint to define primary key for a table.

?? A column, defined as a primary key, cannot have duplicate values across all records and cannot have a null value.

Syntax : ColumnName datatype (size) primary key ??

Example :

create table Account (ano char(3) PRIMARY KEY, Balance number(9), Branch varchar2(10));

A primary key constraint is combination of UNIQUE constraint and NOT NULL constraint.

A table cannot have more than one primary key.

?? If multiple columns need to be defined as primary key column, then only table level definition is applicable.

Maximum 16 columns can be combined as a composite primary key in a table. **Foreign Key Constraint**

A foreign key constraint is also called referential integrity constraint, is specified between two tables.

This constraint is used to ensure consistency among records of the two tables.

The table, in which a foreign key is defined, is called a foreign table, detail table or child table.

The table, of which primary key or unique key is defined, is called a primary table, master table or parent table.

Restriction on child table :

✓ Child table contains a foreign key. And, it is related to master table.

✓ Insert or update operation involving value of foreign key is not allowed, if corresponding value does not exist in a master table.

Restriction on master table :

✓ Master table contains a primary key, which is referred by foreign key in child table.

✓ Delete or update operation on records in master table are not allowed, if corresponding records are present in child table.

Syntax : ColumnName datatype (size) REFERENCES TableName (ColumnName)

Example :

```
create table Account (ano char(3) , Balance number(9),  
Branch varchar2(10) REFERENCES branch_detail(branch));
```

Master table must be exist before creating child table.

Describe the following SQL functions.

SQL Function	Description	SQL Query Example
<i>Numeric function</i>		
Abs(n)	Returns the absolute value of n.	Select Abs(-15) from dual; O/P : 15
Power (m,n)	Returns m raised to n th power.	Select power(3,2) from dual; O/P : 9
Round (n,m)	Returns n rounded to m places the right of decimal point.	Select round(15.91,1) from dual; o/p : 15.9
Sqrt(n)	Returns square root of n.	Select sqrt(25) from dual; O/P : 5
Exp(n)	Returns e raised to the n th power, e=2.17828183.	Select exp(1) from dual; O/P : 1
Gretest()	Returns the greatest value in a list of values.	Select greatest(10, 20, 30) from dual; O/P : 30

Least ()	Returns the least value in a list of values.	Select least(10,20,30) from dual; O/P : 10
----------	--	---

Mod(n,m)	Returns remainder of n divided by m.	Select mod(10,2) from dual; O/P : 0
Ceil(n)	Returns the smallest integer value that is greater than or equal to a number.	Select ceil(24.8) from dual; O/P : 25
ASCII(x)	Returns ASCII value of character.	Select ascii('A') from dual; O/P : 97
Concat()	Concatenates two strings.	Select concat('great','DIET') from dual; O/P : greatDIET
Initcap ()	Changes the first letter of a word in to capital.	Select initcap('diet') from dual; O/P : Diet
Instr ()	Returns a location within the string where search patterns begins.	Select instr('this is test','is') from dual; O/P : 3
Length ()	Returns the number of character in x.	Select length('DIET') from dual; O/P : 4
Lower()	Converts the string to lower case.	Select lower('DIET') from dual; O/P : diet
Upper()	Converts the string to upper case.	Select upper('diet') from dual; O/P : DIET
Lpad()	Pads x with spaces to left to bring the total length of the string up to width characters.	Select lpad('abc',9,'>') from dual; O/P : >>>>>abc
Rpad()	Pads x with spaces to right to bring the total length of the string up to width characters.	Select rpad('abc',9,'>') from dual; O/P : abc>>>>>
Ltrim()	Trim characters from the left of x.	Select ltrim('sumita','usae') from dual; O/P : mita
Rtrim()	Trim characters from the right of x.	Select rtrim('sumita','tab') from dual; O/P : sumi
Replace()	Looks for the string and replace the string every time it occurs.	Select replace('this is college','is','may be') from dual; O/P : thmay be may be college
Substr()	Returns the part of string	Select substr('this is college',6,7) from dual; O/P : is my c
Vsize()	Returns storage size of string in oracle.	Select vsize('abc') from dual; O/P : 3

Miscellaneous function

Uid	Returns integer value corresponding to the UserId.	Select uid from dual; O/P : 618
User	Returns the name of user.	Select user from dual; O/P : admin
Avg(x)	Returns the average value.	Select avg(salary) from employee;
Count(x)	Returns the number of row return by a query.	Select count(deptno) from employee; Returns the number of rows in the table <i>List total number of Employees.</i> SELECT COUNT (*) FROM Employee_Manager; List total number of Employees who have been assigned a Manager. SELECT COUNT (Manager_ID) FROM Employee_Manager; Count(*) = No of rows Count(Column Name) = No. of rows that do not have NULL Value
Max(x)	Returns the maximum value of x.	Select max(salary) from employee;
Min(x)	Returns the minimum value of x.	Select min(salary) from employee;
Median(x)	Returns the median value of x.	Select median(salary) from employee;
Sum(x)	Returns the sum of x.	Select sum(salary) from employee;
Stddev(x)	Returns standard deviation of x.	Select stddev(salary) from employee;
Variance(x)	Returns variance of x.	Select variance(salary) from employee;

Date functions & conversion function

To_char	Takes date data type and returns character string according to acceptable format.	SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD') FROM dual; O/P : 2012-07-19
To_date	Converts x string to date time	Select to_date('1-jan-2011', 'dd-mon-yyyy') from dual; O/P : 1-jan-2011
To_number	Converts character string to number	Select to_number('10') + 20 from dual; O/P : 30
Add_months(x,y)	Gets the result of adding y months to x.	Select add_months('1-jan-2005',1) from dual; O/P : 1-feb-2005
Sysdate	Returns the date of operating system	Select sysdate from dual; O/P : (write today's date)

Last_day	Returns the last day of any month.	Select last_day('01-jan-2005') from dual;
Months_between	Gets the number of months between x and y.	O/P : 31-jan-2005 Select months_between('01-jan-2005','01-feb-2005') from dual; O/P : 1
Next_day	Returns date of the next day following x.	Select next_day('01-jan-2005','saturday') from dual; O/P : 08-jan-2005

Get all distinct Customer Name
SELECT DISTINCT Cust_Last_Name
FROM Customer_Details;

List all Account_No where Total_Available_Balance_in_Dollars is atleast \$10000.00
SELECT Account_No
FROM Customer_Transaction
WHERE Total_Available_Balance_in_Dollars >= 10000.00;

List all Cust_ID, Cust_Last_Name where Account_type is 'Savings' and Bank_Branch is 'Capital Bank'.
SELECT Cust_ID, Cust_Last_Name
FROM Customer_Details
WHERE Account_Type = 'Savings' AND Bank_Branch = 'Capital Bank';

List all Cust_ID, Cust_Last_Name where neither Account_type is 'Savings' and nor Bank_Branch is 'Capital Bank'
SELECT Cust_ID, Cust_Last_Name
FROM Customer_Details
WHERE NOT Account_Type = 'Savings' AND
NOT Bank_Branch = 'Capital Bank';

Logical operator: AND, OR, and NOT

List all Cust_ID, Cust_Last_Name where either Account_type is 'Savings' or Bank_Branch is 'Capital Bank'.

SELECT Cust_ID, Cust_Last_Name
FROM Customer_Details
WHERE Account_Type = 'Savings' OR Bank_Branch = 'Capital Bank';

Retrieval using BETWEEN

List all Account_Nos with balance in the range \$10000.00 to \$20000.00.
SELECT Account_No
FROM Customer_Transaction
WHERE Total_Available_Balance_in_Dollars >= 10000.00
AND Total_Available_Balance_in_Dollars <= 20000.00;

or

SELECT Account_No
FROM Customer_Transaction

WHERE Total_Available_Balance_in_Dollars
BETWEEN 10000.00 **AND** 20000.00

Retrieval using IN

List all customers who have account in Capital Bank or Indus Bank.

```
SELECT Cust_ID
FROM Customer_Details
WHERE Bank_Branch = 'Capital Bank'
OR Bank_Branch = 'Indus Bank';
Or
SELECT Cust_ID
FROM Customer_Details WHERE Bank_Branch IN ('Capital Bank', 'Indus Bank');
```

LIKE predicate uses two symbols for pattern matching.

_ (Underscore)

% (Percentage)

Underscore can be used for single character matching.

Percentage sign can be used for multiple character matching.

List all Accounts where the Bank_Branch begins with a 'C' and has
'a' as the second character

```
SELECT Cust_ID, Cust_Last_Name, Account_No
FROM Customer_Details
WHERE Bank_Branch LIKE 'Ca%';
```

List all Accounts where the Bank_Branch column has 'a' as the
second character.

```
SELECT Cust_ID, Cust_Last_Name, Account_No
FROM Customer_Details
WHERE Bank_Branch LIKE '_a%';
```

searches for employees with the pattern A_B in their name:

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%A\_B%' ESCAPE '\';
```

The **ESCAPE** clause identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (_). This causes Oracle to interpret the underscore literally, rather than as a special pattern matching character.

Retrieval using IS NULL and NOT NULL

List employees who have not been assigned a Manager yet.

```
SELECT Employee_ID  
FROM Employee_Manager  
WHERE Manager_ID IS NULL;
```

List employees who have been assigned to some Manager.

```
SELECT Employee_ID  
FROM Employee_Manager  
WHERE Manager_ID IS NOT NULL;
```

Sorting your results (ORDER BY)

List the customers account numbers and their account balances, in the increasing order of the balance

```
SELECT Account_No, Total_Available_Balance_in_Dollars  
FROM Customer_Transaction  
ORDER BY Total_Available_Balance_in_Dollars;  
• by default the order is ASCENDING
```

List the customers and their account numbers in the decreasing order of the account numbers.

```
SELECT Cust_Last_Name, Cust_First_Name, Account_No  
FROM Customer_Details  
ORDER BY Account_NO DESC
```

or

```
ORDER BY 3 DESC;
```

List the customers and their account numbers in the decreasing order of the Customer Last Name and increasing order of account numbers.

```
SELECT Cust_Last_Name, Cust_First_Name, Account_No  
FROM Customer_Details  
ORDER BY Cust_Last_Name DESC, Account_No;
```

Or

```
SELECT Cust_Last_Name, Cust_First_Name, Account_No  
FROM Customer_Details  
ORDER BY 1 DESC, 3;
```

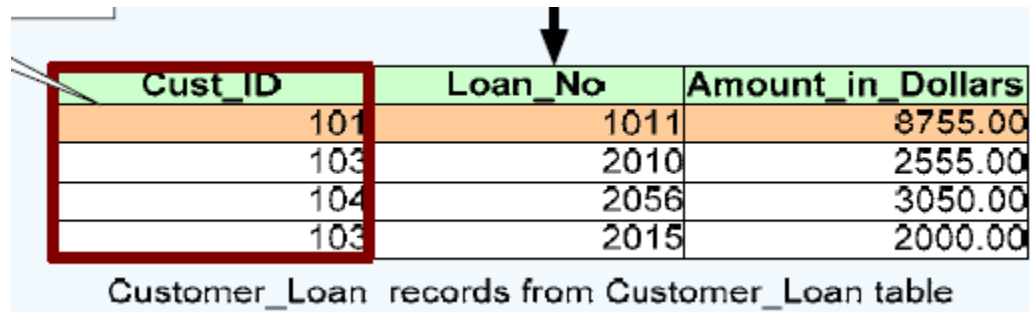
SQL - Using GROUP BY

- Related rows can be grouped together by **GROUP BY** clause by specifying a column as a grouping column.
- **GROUP BY** is associated with an aggregate function
- To retrieve the total loan-amount of all loans taken by each Customer.

```

SELECT Cust_ID, SUM(Amount_in_Dollars)
FROM Customer_Loan
GROUP BY Cust_ID;

```



Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Customer_Loan records from Customer_Loan table

Query Results

Cust_ID	Sum(Amount_in_Dollars)
101	8755.00
103	4555.00
104	3050.00

To retrieve Number of Employees in each Department

```

SELECT Department, COUNT (Employee_ID)
FROM Employee_Manager
GROUP BY Department

```

Employee_ID	Employee_Last_Name	Department	Grade	Manager_ID
2345	Atherton	HR	1	NULL
3556	George	Finance	1	NULL
3620	Jackson	Design	1	NULL
22789	Stevenson	HR	2	2345
23456	Smith	Finance	2	3556
30456	Langer	HR	3	2345
31234	Frost	Finance	3	3556
32345	Austen	Design	2	3620

Department	Count(Employee_ID)
HR	3
Finance	3
Design	2

Retrieval using HAVING

- Used to specify condition on group
- List all customers who are having loans greater than 4000

```

Select Cust_ID,SUM(Amount_in_Dollars)
From Customer_Loan

```

Group By Cust_ID Having SUM(Amount_in_Dollars) > 4000.00;

Use customer table above

Query Results ▼	
Cust_ID	Sum(Amount_in_Dollars)
101	8755.00
103	4555.00

Can you identify any error...?

Select Cust_ID,SUM(Amount_in_Dollars)
From Customer_Loan
Group By Cust_ID **Having** **LOAN_NO** > 4000.00;

Ans: The Having condition has to be based on some column that appears in the select list

SET Operations

Retrieval using UNION

List all the customer who has either Fixed Deposit or Loan or Both

SELECT Cust_ID
FROM Customer_Fixed_Deposit
UNION
SELECT Cust_ID
FROM Customer_Loan;

Cust_ID		Cust_ID
101		101
103		103
104		104
103		

Query Results

101
103
104

Union All

SELECT Cust_ID **FROM** Customer_Fixed_Deposit
UNION ALL
SELECT Cust_ID **FROM** Customer_Loan;

Query Results

101
103
104
103
101
103
104

Union - Restrictions

- The SELECT statements must contain the **same number of columns**
- Data type
 - Each column in the first table must be the same as the **data type** of the corresponding column in the second table.
 - Data width and column name can differ
- Neither of the two tables can be sorted with the **ORDER BY** clause.
- **Combined query results can be sorted**

Retrieval using INTERSECT

List all the customer who have both Fixed Deposit and Loan.

```
SELECT Cust_ID
FROM Customer_Fixed_Deposit
INTERSECT
SELECT Cust_ID
FROM Customer_Loan;
```

Minus

- Get All the Customer who have not taken loan

```
Select Cust_ID
from Customer_details
MINUS
Select Cust_Id
from Customer_loan;
```

Types of Join

Cross JOIN or Cartesian Product

This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECT column-name-list
from table-name1
CROSS JOIN
table-name2;
```

Cross JOIN query will be,

```
SELECT *
from class,
cross JOIN class_info;
```

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.

Inner Join Syntax is,

```
SELECT column-name-list
from table-name1
INNER JOIN
table-name2
WHERE table-name1.column-name = table-name2.column-name;
```

Example of Inner JOIN

The **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

The **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

```
SELECT * from class, class_info where class.id = class_info.id;
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI

3	alex	3	CHENNAI
---	------	---	---------

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

Natural Join Syntax is,

```
SELECT *
from table-name1
NATURAL JOIN
table-name2;
```

Example of Natural JOIN

Natural join query will be,

```
SELECT * from class NATURAL JOIN class_info;
```

The result table will look like,

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

In the above example, both the tables being joined have ID column(same name and same datatype), hence the records for which value of ID matches in both the tables will be the result of Natural Join of these two tables.

Outer JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
 - Right Outer Join
 - Full Outer Join
-

Left Outer Join

The left outer join returns a result table with the **matched data** of two tables then remaining rows of the **left** table and null for the **right** table's column.

Left Outer Join syntax is,

```
SELECT column-name-list
from table-name1
LEFT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Left outer Join Syntax for **Oracle** is,

```
select column-name-list
from table-name1,
table-name2
on table-name1.column-name = table-name2.column-name(+);
```

Example of Left Outer Join

Left Outer Join query will be,

```
SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id=class_info.id);
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI

3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null

Right Outer Join

The right outer join returns a result table with the **matched data** of two tables then remaining rows of the **right table** and null for the **left** table's columns.

Right Outer Join Syntax is,

```
select column-name-list
from table-name1
RIGHT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Right outer Join Syntax for **Oracle** is,

```
select column-name-list
from table-name1,
table-name2
on table-name1.column-name(+) = table-name2.column-name;
```

Example of Right Outer Join

Full Outer Join

The full outer join returns a result table with the **matched data** of two table then remaining rows of both **left**table and then the **right** table.

Full Outer Join Syntax is,

```
select column-name-list
from table-name1
FULL OUTER JOIN
```

```
table-name2
```

```
on table-name1.column-name = table-name2.column-name;
```

Example of Full outer join is,

Full Outer Join query will be like,

```
SELECT * FROM class FULL OUTER JOIN class_info on (class.id=class_info.id);
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null
Null	null	7	NOIDA
Null	null	8	PANIPAT

Self join-Joining a table with itself

When you wish to join a table with itself based on some criteria, use the concept of synonyms. Treat the table as two different tables by giving synonyms

To list all the Employees along with their Managers

Select Emp.Employee_ID as "Employee ID",
Emp.Employee_Last_Name as "Employee Last Name",

Manager.Manager_Id as "Manager ID",

Manager.Employee_Last_Name as "Manager Last Name",
From employee_Manager Emp , employee_Manager Manager
Manager.Employee_ID;

Where Emp.Manager_ID =

Employee_ID	Employee_Last_Name	Manager_ID
2345	Atherton	NULL
3556	George	NULL
3620	Jackson	NULL
22789	Stevenson	2345
23456	Smith	3556
30456	Langer	2345
31234	Frost	3556
32345	Austen	3620

Query Results

Employee ID	Employee Last Name	Manager ID	Manager Last Name
22789	Stevenson	2345	Atherton
23456	Smith	3556	George
30456	Langer	2345	Atherton
31234	Frost	3556	George
32345	Austen	3620	Jackson

Subquery : (Query inside query)

A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Independent sub-queries

- Inner query is independent of outer query.
- Inner query is executed first and the results are stored.
- Outer query then runs on the stored results.

These are queries where there are two parts to the query. We need to collect one type of information based on which other set of information has to be retrieved from the table.

For e.g :

Select all sales reps who have a higher quota than sales rep 101.

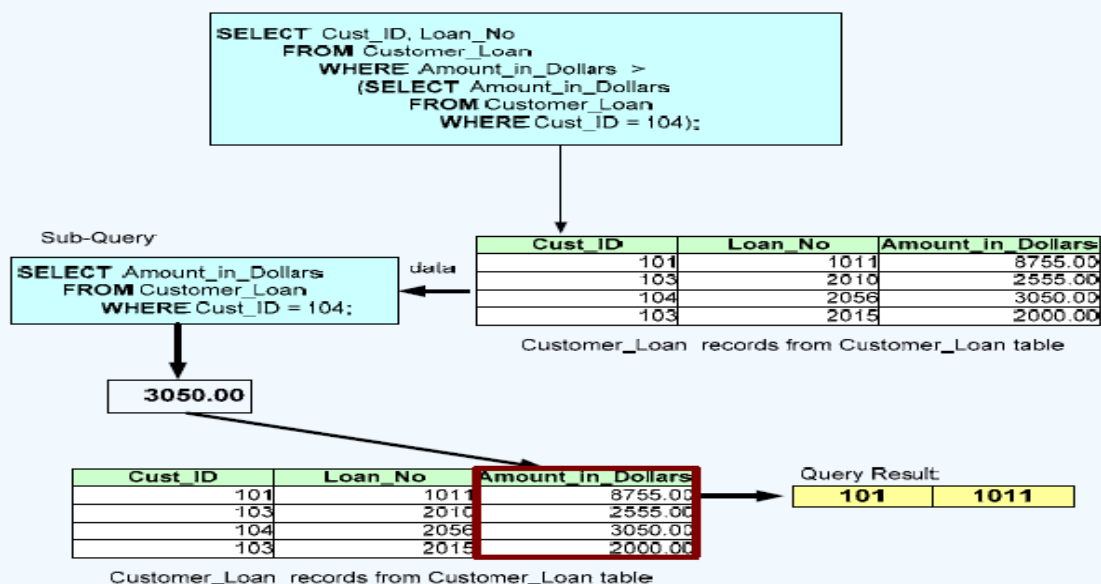
We need to analyze this query and understand how to break it into sub problems

1. *First we need to find out what is the quota of sales rep 101*
2. *Based on this info, we need to select sales reps who have a higher quota than this value*
3. *So, the inner query will find the quota of sales rep 101 and the outer query will extract sales reps exceeding this quota value. The solution would look like:*

```
SELECT Rep
FROM SalesReps
WHERE Quota >
(SELECT Quota
FROM SalesReps
WHERE Empl_Num = 101;
```

To list the Cust_ID and Loan_No for all Customers who have taken a loan of amount greater than the loan amount of Customer (Cust_ID = 104).

Sub Query (Contd...)



List customer names of all customers who have taken a loan > \$3000.00.

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
FROM Customer_Details
```

```
WHERE Cust_ID
IN
( SELECT Cust_ID
FROM Customer_Loan
WHERE Amount_in_Dollars > 3000.00);
```

List customer names of all customers who have the same Account_type as Customer 'Jones Simon' .

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
FROM Customer_Details
WHERE Account_Type
=
( SELECT Account_Type
FROM Customer_Details
WHERE Cust_Last_Name = 'Jones'
AND Cust_First_Name = 'Simon');
```

List customer names of all customers who do not have a Fixed Deposit.

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
FROM Customer_Details
WHERE Cust_ID
NOT IN
( SELECT Cust_ID
FROM Customer_Fixed_Deposit);
```

List customer names of all customers who have either a Fixed Deposit or a loan but not both at any of Bank Branches. The list includes customers who have no fixed deposit and loan at any of the bank branches.

```
SELECT Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
FROM Customer_Details
WHERE Cust_ID
NOT IN
( SELECT Cust_ID
FROM Customer_Loan
WHERE Cust_ID
IN
(SELECT Cust_ID
FROM Customer_Fixed_Deposit ));
```

Correlated Sub Queries

- You can refer to the table in the FROM clause of the outer query in the

inner query using Correlated sub-queries.

- The inner query is executed separately for each row of the outer query. (i.e. In Co-Related Sub-queries, SQL performs a sub-query over and over again – once for each row of the main query.)

To list all Customers who have a fixed deposit of amount less than the sum of all their loans.

```
Select Cust_Id, Cust_Last_Name, cust_Mid_Name, cust_First_Name
From Customer_fixed_Deposit
Where amount_in_dollars
<
(Select sum(amount_in_dollars)
From Customer_Loan
Where Customer_Loan.Cust_Id = Customer_Fixed_Deposit.Cust_ID);
```

List customer IDs of all customers who have both a Fixed Deposit and a loan at any of Bank Branches

```
SELECT Cust_ID
FROM Customer_Details
WHERE Cust_ID
IN
(SELECT Cust_ID
FROM Customer_Loan
WHERE Customer_Loan.Cust_ID = Customer_Details.Cust_ID)
AND Cust_ID IN
(SELECT Cust_ID
FROM Customer_Fixed_Deposit
WHERE Customer_Fixed_Deposit.Cust_ID = Customer_Details.Cust_ID);
```

Exists and not exists

List customer IDs of all customers who have both a Fixed Deposit and a loan at any of Bank Branches

Exists check for the existence of a situation/condition.

```
SELECT Cust_ID
FROM Customer_Fixed_Deposit
WHERE EXISTS
(SELECT *
FROM Customer_Loan
WHERE Customer_Loan.Cust_ID = Customer_Fixed_Deposit.Cust_ID);
```

List all Customers who don't have a single Fixed Deposit over \$3000.00.

```
SELECT Cust_ID, Cust_Last_Name, Cust_Mid_Name, Cust_First_Name
FROM Customer_Details S
```



```
WHERE  
NOT EXISTS  
(SELECT *  
FROM Customer_Fixed_Deposit O  
WHERE O.Amount_in_Dollars > 3000.00 AND O.Cust_ID = S.Cust_ID);
```

“All” Construct Compares Value To Every Value Returned by the Sub Query.
Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch_name  
from branch  
where assets > all  
(select assets  
from branch  
where branch_city = 'Brooklyn')
```

ANY:-> Compares Value to Each Value Returned by the Sub Query

Means More Than The Minimum Value in the List.
Means Less Than The Maximum Value in the List.

Note : 'Some' is also used insted of ANY.

While using all

```
SELECT empno, sal FROM emp WHERE sal > ALL (2000, 3000, 4000);
```

EMPNO	SAL
7839	5000

It will return result equivalent to query:

```
SELECT empno, sal FROM emp WHERE sal > 2000 AND sal > 3000 AND sal > 4000;
```

While using any

```
SELECT empno, sal FROM emp WHERE sal > ANY (2000, 3000, 4000);
```

EMPNO	SAL
7566	2975
7698	2850
7782	2450
7788	3000
7839	5000
7902	3000

It will return result equivalent to query:

```
SELECT empno, sal FROM emp WHERE sal > 2000 OR sal > 3000 OR sal > 4000;
```

Views

What is a view?

- A view is a kind of “virtual table”

Views are tables whose contents are taken or derived from other tables.

To the user, the view appears like a table with columns and rows

But in reality, the view doesn't exist in the database as a stored set of values

The rows and columns that we find inside a view are actually the results generated by a query that defines the view

View is like a window through which we see a limited region of the actual table

The table from where the actual data is obtained is called the source table

```
CREATE VIEW ViewCustomerDetails  
AS SELECT *  
FROM Customer_Details;
```

You must have permission on the table referred in the view to successfully create the view

Can assign a new name to each column chosen in the view

Only names can be different. The data type etc remain the same as the source table because, the values are after all going to be derived from that table

If no names are specified for columns, the same name as used in the source table is used

Assigning names to columns

```
Create view vwCustDetails (CustCode,CustLname,CustFName)  
AS Select Cust_Id,Cust_Last_Name,Cust_First_Name  
From Customer_details
```

Types of views

- Horizontal views
- Vertical views
- Row/column subset views
- Grouped views
- Joined views

Horizontal views

Horizontal view restricts a user's access to only selected rows of a table.

```
CREATE VIEW view_cust AS  
  
    SELECT *  
  
    FROM Customer_Details  
  
    WHERE Cust_ID in (101,102,103);
```

Vertical views

- A view which selects only few columns of a table:
- Vertical view restricts a user's access to only certain columns of a table

```
CREATE VIEW view_cust AS  
  
    SELECT Cust_ID, Account_No, Account_Type  
  
    FROM Customer_Details;
```

Row/column subset views

```
CREATE VIEW View_Cust_VertHor  
AS SELECT Cust_Id,Account_No,Account_Type  
FROM Customer_Details  
WHERE CUST_ID IN (101,102,103);
```

There could be a combination of these two concepts where a view selects a subset of rows and columns

Views with Group By clause

- The query contains a group by clause

```
CREATE VIEW View_GroupBY(Dept,NoofEmp)  
AS SELECT Department, count(Employee_ID)  
FROM Employee_Manager  
GROUP BY Department;
```

Views with Joins

- Created by specifying a two-table or three-table query in the view creation command

```
Create view View_Cust_Join as  
Select a.Cust_Id,b.Cust_First_Name,b.Cust_Last_Name,Amount_in_dollars  
from Customer_loan a, customer_details b  
where a.cust_id = b.cust_id;
```

Updating a VIEW A view can be modified by the DML command.

```
CREATE VIEW View_Cust  
AS SELECT *  
FROM Customer_Details  
WHERE CUST_ID IN (101,102,103);
```

--Insert Statement

```
insert into view_cust values(103,'Langer','G.','Justin',3421,'Savings',' Global  
Commerce Bank','Langer_Justin@Yahoo.com');
```

--Delete Statement

```
delete view_cust where cust_id=103;
```

--Update Statement

```
Update view_cust  
set Cust_last_name='Smyth'  
where cust_id=101;
```

So a view is updatable if :
DISTINCT is not specified in the query used to create the view
The FROM clause specifies only one source table
The select list doesn't contain expressions/calculated columns
The WHERE clause doesn't include a subquery

The query doesn't include a GROUP BY or HAVING

Dropping Views

DROP VIEW <view name>;

DROP VIEW View_Cust

if some other views depend on this view, then if you say

DROP VIEW ViewSupplier CASCADE then this view plus all other views that are based on this view are deleted.

If you specify

DROP VIEW ViewSupplier RESTRICT then this view cannot be deleted if other views depend on it.

Advantages of views

Security: only a limited set of rows/ columns are viewable by certain users

Query simplicity: A view can derive data from many tables. So, subsequently we can use queries on the view as single table queries rather than writing queries against the source tables as multi-table queries

Structural simplicity: views can show only a portion of the table which is relevant to the user there by keeping it simple.

Disadvantages of views

Performance: views based on joins are merely virtual tables. Every time a query is placed against the view, the query representing creation of the view has to be executed . So, complex joins may have to be performed every time a query is placed against the view.

Restrictions: Not all views are updateable

Transaction Control Language(TCL) commands are used to manage transactions in database. These are used to manage the changes made by DML statements

The COMMIT Command:

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for COMMIT command is as follows:

```
COMMIT;
```

The ROLLBACK Command:

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

```
ROLLBACK;  
DELETE FROM CUSTOMERS  
    WHERE AGE = 25;  
SQL> ROLLBACK;
```

The SAVEPOINT Command:

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for SAVEPOINT command is as follows:

```
SAVEPOINT SAVEPOINT_NAME;
```

This command serves only in the creation of a SAVEPOINT among transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as follows:

```
ROLLBACK TO SAVEPOINT_NAME;  
SQL> SAVEPOINT SP1;  
Savepoint created.
```

```
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
```

1 row deleted.

```
SQL> SAVEPOINT SP2;
```

Savepoint created.

```
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
```

1 row deleted.

```
SQL> SAVEPOINT SP3;
```

Savepoint created.

```
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
```

1 row deleted.

Now that the three deletions have taken place, say you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone:

```
SQL> ROLLBACK TO SP2;
```

Rollback complete. SQL> SAVEPOINT SP1;

Savepoint created.

```
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
```

1 row deleted.

```
SQL> SAVEPOINT SP2;
```

Savepoint created.

```
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
```

1 row deleted.

```
SQL> SAVEPOINT SP3;
```

Savepoint created.

```
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
```

1 row deleted.

Now that the three deletions have taken place, say you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone:

```
SQL> ROLLBACK TO SP2;
```

Rollback complete.

DCL Data Control Language(DCL) is used to control privilege in Database.

Give and take permission from user.

GRANT Tables or views

```
GRANT {  
[ALTER[, ]]  
[DELETE[, ]]  
[INDEX[, ]]  
[INSERT[, ]]
```

```
[SELECT[, ]]
[UPDATE [(column-name[,...])][, ]]
| ALL [PRIVILEGES] }
ON [TABLE] {table-name[,...] | view-name[,...]}
TO [AuthID][,...]
[WITH GRANT OPTION]
```

```
GRANT SELECT, INSERT
ON Customer_Details
TO Edwin ;
GRANT ALL PRIVILEGES
ON Customer_Loan
TO JACK ;
GRANT ALL
ON Customer_Loan
TO PUBLIC ;
```

Grant

- With Grant Option

```
GRANT SELECT
ON Customer_Loan
TO EDWIN
With GRANT OPTION;
```

Taking PRIVILIGES away

The syntax of REVOKE command is patterned after GRANT, but with a reverse meaning.

```
REVOKE{
[ALTER[, ]]
[DELETE[, ]]
[INDEX[, ]]
[INSERT[, ]]
[SELECT[, ]]
[UPDATE [(column-name[,...])][, ]]
| ALL [PRIVILEGES] }
ON [TABLE] {table-name[,...] | view-name [,...]}
FROM AuthID[,...]
```

```
REVOKE SELECT, INSERT
ON Customer_Details
FROM Edwin ;
```

```
REVOKE ALL PRIVILEGES
ON Customer_Loan
FROM JACK ;
```

```
REVOKE ALL
ON Customer_Loan
FROM PUBLIC ;
```