Data type:

The oracle has following significance datatypes.
1) NUMBER
2) CHAR
3) VARCHAR
4) VARCHAR2
5) COB
6) NCLOB
7) LONG
8) DATE
9) RAW
10) LONGROW

1) NUMBER

The Oracle NUMBER data type is used to store numeric values that can be positive or negative.

syntax:-

NUMBER ((p [s])]

- The Oracle NUMBER data type has precision (p) and scale (s). Precision is the number of decimal digits and scale is the number of digits to the right of the decimal point. The precision can have ranges from 1 to 38 and scale can have ranges from -84 to 127.

Example:

```
CREATE TABLE Employee (Emp_No NUMBER (5), Emp_No NUMBER (6,2));
```

2) CHAR

Char data type used to store character value of fixed length. The maximum number of character this datatype can hold upto 255 characters. In CHAR datatype data can be padded with spaces.

syntax:-

CHAR (SIZE)

- The size in the bracket determine the numbers of character the cell can hold.

Example:

```
CREATE TABLE Employee (Emp_No NUMBER (5), Emp_Name CHAR (15));
```

3) VARCHAR2

Varchar2 data type is used to store character value of variable length. The maximum number of character this datatype can hold upto 4000 characters. In varchar2 datatype data cannot be padded with spaces.

syntax:-

VARCHAR2 (SIZE)

- the bracket determine the numbers of character the cell can hold.

Example:

```
CREATE TABLE Employee (Emp_No NUMBER (5), Emp_Name VARCHAR2 (15));
```

4) VARCHAR

The VARCHAR datatype is currently same as the VARCHAR2 datatype. In a future the VARCHAR datatype might store variable-length character values. So that, use the VARCHAR2 datatype to store variable-length character strings values.

syntax:-

VARCHAR (SIZE)

- The size in the bracket determine the numbers of character the cell can hold.

Example:

```
CREATE CLOB TABLE Dhunna Employee (Emp_No NUMBER (5), Emp_Name VARCHAR (15));
```

5) CLOB:

CLOB stands for Character Large Object, this data type is used to store character large object (single character sets) data. The maximum number of character this datatype can hold upto 4GB characters.

6) NCLOB:

NCLOB stands for National Character Large Object. This data type is used to store Unicode national character set (multiple character sets) data. The maximum number of character this datatype can hold upto 4GB characters.

7) LONG:

LONG datatype is used to store Variable-length character data. The maximum number of character this datatype can hold upto 2431-1 bytes, or 2GB.

8) DATE:

Date data type is used to store date and time value of fixed length. The standard format is DD-MON-YY as 28-JUL-20. The range of date datatype is from January 1,4712 B.C. to December 31, 4712 A. D.

9) RAW:

RAW datatype is used to store Variable-length binary data. The maximum number of binary data this datatype can hold upto 255 bytes.

10) LONGROW:

LONGRAW datatype is used to store Variable-length binary data. The maximum number of binary data this datatype can hold upto 2431-1 bytes, or 2GB.

ROWID pseudo column

- A Pseudo column which may work like a table column but really it is not a part of a table.
- The ROWID pseudo column returns the address of the row stored in table. You cannot use ROWID as the and primary of a table.. You can use the ROWID pseudo column with the SELECT and WHERE statements.
- ROWID pseudo column values are not stored in the database.
- You cannot insert, update, or delete a value in the ROWID pseudo column.

Example:

Write a table selects. statement to display the address for all rows that contain in employees table.
SELECT ROWID, Emp_Name FROM employees;

Uses of ROWID Values:

1) ROWID is the fastest way to access a single row from database.
2) ROWID can display how rows physically stored in table.
3) ROWID is the UNIQUE address for a row in a table.
4) ROWID's address may be in two formats (Restricted or Extended)
5) A ROWID can never change during the life time of its row.
6)When a row is deleted, oracle may reassign its ROWID to the new row that is
inserted in table.
7) The ROWID Can never be inserted, updated and deleted manually.
8) The ROWID Pseudo column can be used with select and where clauses

DUAL table

What is dual table?

Dual table is a small oracle work table, which is consist of one row and one column with value x. Oracle provides one dummy table it also called dual table. Dual table will use to calculate arithmetic operation and to display the date with select statement. To see the structure of dual table you should run DESC DUAL; command on SQL prompt. The structure of dual table is as follow.

```
SQL> DESC DUAL;
Name            Null            Type
Dummy                     varchar2  (10)
```

Examples:

1. Select 5 * 5 from dual;

Output:-

```
                    5 * 5
                     25
```

2. Select sysdate from dual;
Output:

```
                   sysdate
                   28\jul\20
```

Sysdate is used to display date and time in oracle

DATE Functions

The value in DATE datatype column is stored in specific default format. This default format is 'DD-MON-YY HH: MI: SS'. To manipulate and extract values from the date column of table oracle provides some date functions. Oracle has following most useful date functions as follows.

1) SYSDATE
2) SYSTIMESTAMP
3) TO_CHAR
4) TRUNC
5) ROUND
6) NEXT DAY
7) LAST DAY
8) MONTHS BETWEEN
9) ADD_MONTHS

1) SYSDATE

The SYSDATE function is used to retrieve the current date as well as time that is set in the operating system on which the database installed.

Syntax:

```
SYSDATE;
```

Example:

```
Select SYSDATE from DUAL;
```

Output:

```
SYSDATE
01-JAN-21
```

2) SYSTIMESTAMP

The seconds SYSTIMESTAMP and time zone function of the operating is used to system retrieves on which the system the database date, including installed fractional.

Syntax:-

```
SYSTIMESTAMP;
```

Example:-

```
Select SYSTIMESTAMP from DUAL;
```

Output:-

```
----------------SYSTIMESTAMP
01-JAN-21 12: 22: 32.871000000 PM +05: 30
```

3) TO_CHAR

The TO_CHAR () is used to converts a value of Date datatype to character data type.

Syntax:-

```
TO_CHAR (< date value> (fmt])
Fmt:-specifies in which format the DATE has to appear.
```

Example:-

```
Select TO_CHAR(SYSDATE, ' DD-MM-YY’) " New Date " from DUAL;
```

Output:

```
  New Date
  28-07-20
```

4) TRUNC

TRUNC () function is used to returns a date value truncated to a specific formate

Syntax:-

```
TRUNC (< date value> [, fmt))
Fmt:-It refers to in which the date value will be truncated.
```

Example:-

```
Select TRUNC(SYSDATE, ' MM') " New Truncated Date Format" from DUAL;
```

Output:-

```
New Truncated Date Format
        28-07-20
```

5) ROUND

The ROUND () function is used to retrieves the rounded date according to the entity specified by the format model. ROUND () function works according to the rules of the Gregorian calendar.

Syntax:-

```
ROUND (< date value> [, fmt])
Fmt:-It refers to in which the date value will be rounded.
```

Example:-

```
Select ROUND(TO_DATE ('16 -SEP-2020’),' MONTH ')" New Rounded Month ",     ROUND(TO_DATE ('16 -SEP-2020'), '
YEAR') " New Rounded Year" FROM DUAL;
```

Output:

```
New Rounded Month-----------------New Rounded Month
    01-OCT-2020                  01-JAN-2021
```

6) NEXT DAY

NEXT_DAY () function is used to returns the date of 1 "' weekday named by char that is after the date.

Syntax:

NEXT_DAY (date, char)
char:-It must be day of the week.

Example:-

Select NEXT_DAY ('11-july-20 ',' sunday ')" Next day " from DUAL;

Output:

```
        Next day
        18-july-20
```

7) LAST DAY

LAST_DAY () function is used to returns the last date of the month specified in the functions.

Syntax:-

LAST_DAY (date)

Example:-

Select SYSDATE, LAST_DAY (SYSDATE)" Last day " from Dual;

Output:

```
           Last day
01-07-2020 31-07-2020
```

8) MONTHS BETWEEN

MONTHS BETWEEN () function is used to returns no. of months between datel and date2.

Syntax:-

MONTHS_BETWEEN (datel, date2)

Example:-

Select MONTHS_BETWEEN ('01 -JUL-20', ' 01-JUN-20') " Months" from DUAL;

Output:

```
Months
  1
```

9) ADD_MONTHS

ADD_MONTHS () function is used to add the no. of months specified in the function.

Syntax:-

ADD_MONTHS (d, n)

Example:-

Select ADD_MONTHS (SYSDATE, 2) " Add Months" from DUAL;

Output:

```
Add Months
10-03-2020
```

SQL INDEX

The Index in SQL is a special table used to speed up the searching of the data in the database tables. It also retrieves a vast amount of data from the tables frequently. The INDEX requires its own space in the hard disk.
The index concept in SQL is same as the index concept in the novel or a book.
It is the best SQL technique for improving the performance of queries. The drawback of using indexes is that they slow down the execution time of UPDATE and INSERT statements. But they have one advantage also as they speed up the execution time of SELECT and WHERE statements.
In SQL, an Index is created on the fields of the tables. We can easily build one or more indexes on a table. The creation and deletion of the Index do not affect the data of the database.
In this article, you will learn how to create, alter, and remove an index in the SQL database.

Why SQL Index?

The following reasons tell why Index is necessary in SQL:

- SQL Indexes can search the information of the large database quickly.
- This concept is a quick process for those columns, including different values.
- This data structure sorts the data values of columns (fields) either in ascending or descending order. And then, it assigns the entry for each value.
- Each Index table contains only two columns. The first column is row_id, and the other is indexed-column.
- When indexes are used with smaller tables, the performance of the index may not be recognized.

Create an INDEX

In SQL, we can easily create the Index using the following CREATE Statement:

```
CREATE INDEX Index_Name ON Table_Name ( Column_Name);
```

Here, **Index_Name** is the name of that index that we want to create, and **Table_Name** is the name of the table on which the index is to be created. The **Column_Name** represents the name of the column on which index is to be applied.

If we want to create an index on the combination of two or more columns, then the following syntax can be used in SQL:

```
CREATE INDEX Index_Name ON Table_Name ( column_name1, column_name2, .... , column_nameN);
```

Example for creating an Index in SQL:

Let's take an Employee table:

Emp_Id	Emp_Name	Emp_Salary	Emp_City	Emp_State
1001	Akshay	20000	Noida	U.P

1002	Ram	35000	Jaipur	Rajasthan
1003	Shyam	25000	Gurgaon	Haryana
1004	Yatin	30000	Lucknow	U.P

The following SQL query creates an Index **'Index_state'** on **the Emp_State** column of the **Employee** table.

```
CREATE INDEX index_state ON Employee (Emp_State);
```

Suppose we want to create an index on the combination of the **Emp_city** and the **Emp_State** column of the above **Employee** table. For this, we have to use the following query:

```
CREATE INDEX index_city_State ON Employee (Emp_City, Emp_State);
```

Create UNIQUE INDEX

Unique Index is the same as the Primary key in SQL. The unique index does not allow selecting those columns which contain duplicate values.

This index is the best way to maintain the data integrity of the SQL tables.

Syntax for creating the Unique Index is as follows:

```
CREATE UNIQUE INDEX Index_Name ON Table_Name ( Column_Name);
```

Example for creating a Unique Index in SQL:

Let's take the above Employee table. The following SQL query creates the unique index **index_salary** on the **Emp_Salary** column of the **Employee** table.

```
CREATE UNIQUE INDEX index_salary ON Employee (Emp_Salary);
```

Rename an INDEX

We can easily rename the index of the table in the relational database using the ALTER command.

Syntax:

```
ALTER INDEX old_Index_Name RENAME TO new_Index_Name;
```

Example for Renaming the Index in SQL:

The following SQL query renames the index **'index_Salary'** to **'index_Employee_Salary'** of the above Employee table:

```
ALTER INDEX index_Salary RENAME TO index_Employee_Salary;
```

Remove an INDEX

An Index of the table can be easily removed from the SQL database using the DROP command. If you want to delete an index from the data dictionary, you must be the owner of the database or have the privileges for removing it.

Syntaxes for Removing an Index in relational databases are as follows:

In Oracle database:

```
DROP INDEX Index_Name;
```

In MySQL database:

```
ALTER TABLE Table_Name DROP INDEX Index_Name;
```

In Ms-Access database:

```
DROP INDEX Index_Name ON Table_Name;
```

In SQL Server Database:

```
DROP INDEX Table_Name.Index_Name;
```

Example for removing an Index in SQL:

Suppose we want to remove the above **'index_Salary'** from the SQL database. For this, we have to use the following SQL query:

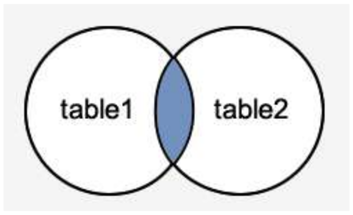
```
DROP INDEX index_salary;
```

SQL JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- If you want to access more than one table through a select statement.
- If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.
- The joining of two or more tables is based on common field between them.
- SQL INNER JOIN also known as simple join is the most common type of join.
- SQL defines Three types of JOIN :
  1. Inner join
  2. Outer join
  3. Cross join

INNER JOIN

- This JOIN returns all records from multiple tables that satisfy the specified join condition. It is the simple and most popular form of join and assumes as a default join. If we omit the INNER keyword with the JOIN query, we will get the same output.
- The following visual representation explains how INNER JOIN returns the matching records from table1 and table2:



INNER JOIN Syntax

The following syntax illustrates the use of INNER JOIN in SQL Server:

```
SELECT columns
FROM table1
INNER JOIN table2 ON condition1
INNER JOIN table3 ON condition2
```

INNER JOIN Example

Let us first create two tables "Student" and "Fee" using the following statement:

```
CREATE TABLE Student (
id number(5) PRIMARY KEY,
admission_no char(10),
first_name char(10),
last_name char(10),
age number(3),
city char(10));

CREATE TABLE Fee (
admission_no char(10),
course char(10),
amount_paid number(7));
```

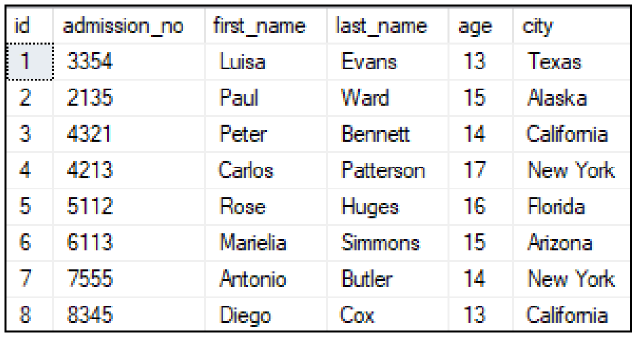
Next, we will insert some records into these tables using the below statements:

```
INSERT INTO Student VALUES(1,3354,'Luisa', 'Evans', 13, 'Texas');
INSERT INTO Student VALUES (2,2135, 'Paul', 'Ward', 15, 'Alaska');
INSERT INTO Student VALUES (4,3321, 'Peter', 'Bennett', 14, 'California');
INSERT INTO Student VALUES (5,4213,'Carlos', 'Patterson', 17, 'New York');
INSERT INTO Student VALUES (6,5112, 'Rose', 'Huges', 16, 'Florida');
INSERT INTO Student VALUES (7,6113, 'Marielia', 'Simmons', 15, 'Arizona');
INSERT INTO Student VALUES (8,7555,'Antonio', 'Butler', 14, 'New York');
INSERT INTO Student VALUES (9,8345, 'Diego', 'Cox', 13, 'California');

INSERT INTO Fee VALUES(3354,'Java', 20000);
INSERT INTO Fee VALUES(7555, 'Android', 22000);
INSERT INTO Fee VALUES(4321, 'Python', 18000);
INSERT INTO Fee VALUES(8345,'SQL', 15000);
INSERT INTO Fee VALUES(5112, 'Machine', 30000);
```

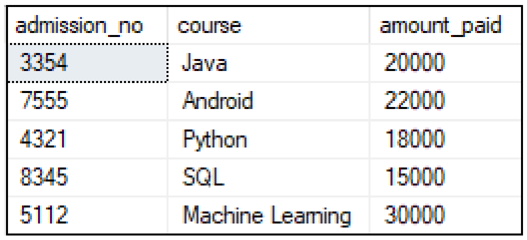
Execute the SELECT statement to verify the records:

Table: Student



id	admission_no	first_name	last_name	age	city
1	3354	Luisa	Evans	13	Texas
2	2135	Paul	Ward	15	Alaska
3	4321	Peter	Bennett	14	California
4	4213	Carlos	Patterson	17	New York
5	5112	Rose	Huges	16	Florida
6	6113	Marielia	Simmons	15	Arizona
7	7555	Antonio	Butler	14	New York
8	8345	Diego	Cox	13	California

Table: Fee



admission_no	course	amount_paid
3354	Java	20000
7555	Android	22000
4321	Python	18000
8345	SQL	15000
5112	Machine Learning	30000

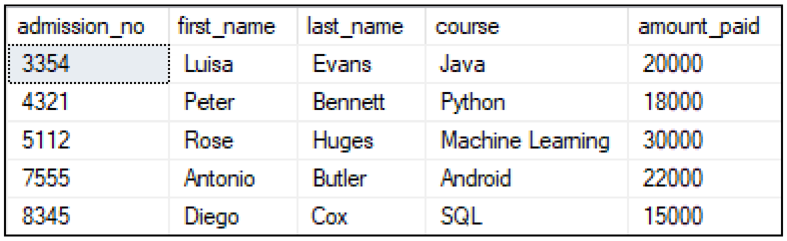
We can demonstrate the INNER JOIN using the following command:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student
INNER JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

OR

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student ,Fee where Student.admission_no = Fee.admission_no;
```

This command gives the below result:



admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
4321	Peter	Bennett	Python	18000
5112	Rose	Huges	Machine Learning	30000
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

In this example, we have used the admission_no column as a join condition to get the data from both tables. Depending on this table, we can see the information of the students who have paid their fee.

OUTER JOIN

OUTER JOIN in SQL Server returns all records from both tables that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.

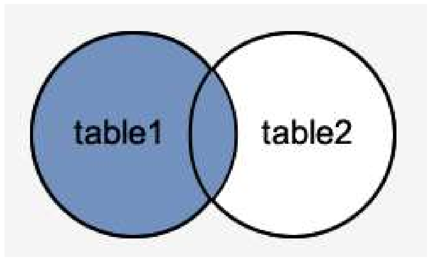
We can categories the OUTER JOIN further into three types:

- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

LEFT OUTER JOIN

The LEFT OUTER JOIN retrieves all the records from the left table and matching rows from the right table. It will return NULL when no matching record is found in the right side table. Since OUTER is an optional keyword, it is also known as LEFT JOIN.

The below visual representation illustrates the LEFT OUTER JOIN:



LEFT OUTER JOIN Syntax

The following syntax illustrates the use of LEFT OUTER JOIN in SQL Server:

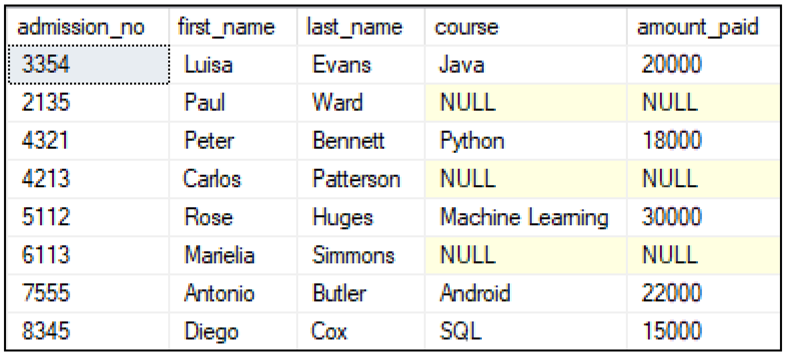
```
SELECT column_lists
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Example

We can demonstrate the LEFT OUTER JOIN using the following command:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student
LEFT OUTER JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:



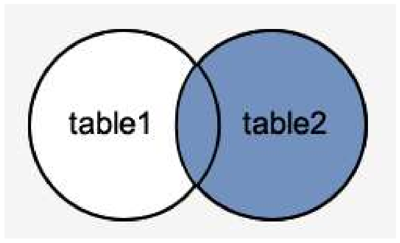
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marielia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

This output shows that the unmatched row's values are replaced with NULLs in the respective columns.

RIGHT OUTER JOIN

The RIGHT OUTER JOIN retrieves all the records from the right-hand table and matched rows from the left-hand table. It will return NULL when no matching record is found in the left-hand table. Since OUTER is an optional keyword, it is also known as RIGHT JOIN.

The below visual representation illustrates the RIGHT OUTER JOIN:



RIGHT OUTER JOIN Syntax

The following syntax illustrates the use of RIGHT OUTER JOIN in SQL Server:

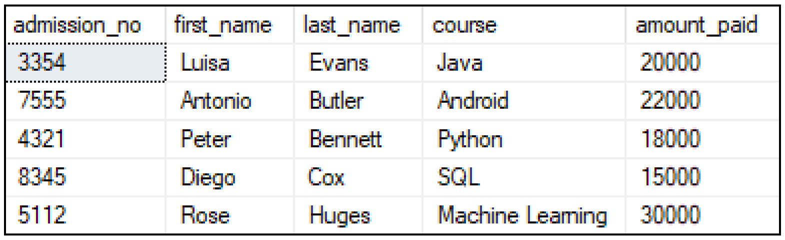
```
SELECT column_lists
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Example

The following example explains how to use the RIGHT OUTER JOIN to get records from both tables:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student
RIGHT OUTER JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:



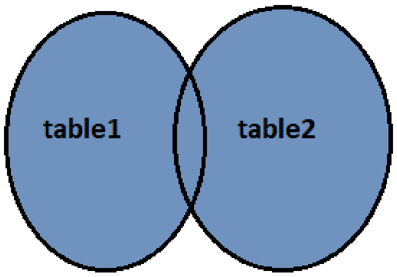
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
7555	Antonio	Butler	Android	22000
4321	Peter	Bennett	Python	18000
8345	Diego	Cox	SQL	15000
5112	Rose	Huges	Machine Learning	30000

In this output, we can see that no column has NULL values because all rows in the Fee table are available in the Student table based on the specified condition.

FULL OUTER JOIN

The FULL OUTER JOIN in SQL Server returns a result that includes all rows from both tables. The columns of the right-hand table return NULL when no matching records are found in the left-hand table. And if no matching records are found in the right-hand table, the left-hand table column returns NULL.

The below visual representation illustrates the FULL OUTER JOIN:



FULL OUTER JOIN Syntax

The following syntax illustrates the use of FULL OUTER JOIN in SQL Server:

```
SELECT column_lists
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

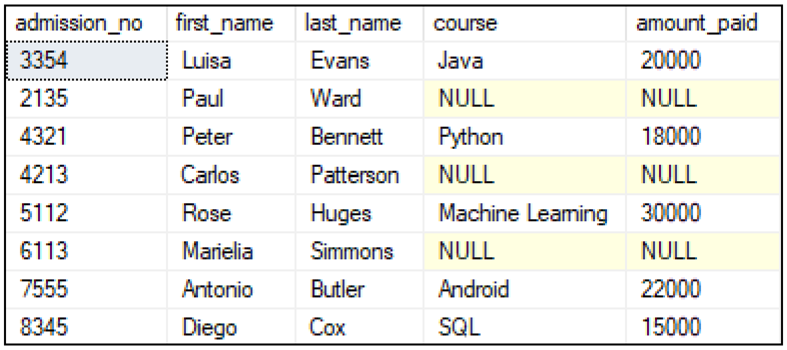
Example

The following example explains how to use the FULL OUTER JOIN to get records from both tables:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student
FULL OUTER JOIN Fee
```

```
ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:



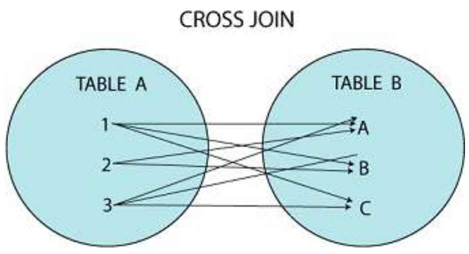
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marielia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

In this output, we can see that the column has NULL values when no matching records are found in the left-hand and right-hand table based on the specified condition.

CROSS JOIN

CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as CARTESIAN JOIN because it produces the Cartesian product of all linked tables. The Cartesian product represents all rows present in the first table multiplied by all rows present in the second table.

The below visual representation illustrates the CROSS JOIN. It will give all the records from table1 and table2 where each row is the combination of rows of both tables:



CROSS JOIN Syntax

The following syntax illustrates the use of CROSS JOIN in SQL Server:

```
SELECT column_lists
FROM table1
CROSS JOIN table2;
```

Example

We can demonstrate the CROSS JOIN using the following command:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
FROM Student
CROSS JOIN Fee;
```

This command gives the result 8*5=40 row because student table have 8 row and fee table have 5 row.

SQL Sub query or Inner query or Nested query

The Subquery or Inner query is an SQL query placed inside another SQL query. It is embedded in the HAVING or WHERE clause of the SQL statements.

Syntax

Query(subquery)

Following are the important rules which must be followed by the SQL Subquery:

1. The SQL subqueries can be used with the following statements along with the SQL expression operators:

1. **SELECT**
2. **UPDATE**
3. **INSERT**
4. **DELETE**

2. The subqueries in SQL are always enclosed in the parenthesis and placed on the right side of the SQL operators.
3. We cannot use the ORDER BY clause in the subquery. But, we can use the GROUP BY clause, which performs the same function as the ORDER BY clause.
4. If the subquery returns more than one record, we have to use the multiple value operators before the Subquery.
5. We can use the BETWEEN operator within the subquery but not with the subquery.

Example

```
create table faculty(
fno number(5) primary key,
fname char(20),
fcity char(20),
phno number(10));

insert into faculty values(101,'Rajesh','Surat',9988776655);
insert into faculty values(102,'Vaibhav','Surat',8899776655);
insert into faculty values(103,'Sonal','Vadodara',7788996655);

create table subject(
sno number(5) primary key,
sname char(20),
lecture number(5),
fno number(5) references faculty(fno));

insert into subject2 values(1,'RDBMS',3,101);
insert into subject2 values(2,'DBMS',4,102);
insert into subject2 values(3,'DBMS',5,103);
insert into subject2 values(4,'RDBMS',3,103);
```

faculty :

fno	fname	fcity	phno
101	Rajesh	Surat	9988776655
102	Vaibhav	Surat	8899776655
103	Sonal	Vadodara	7788996655

Subject:

sno	sname	lecture	fno
1	RDBMS	3	101
2	DBMS	4	102
3	DBMS	5	103
4	RDBMS	3	103

Sub query with SELECT statement

In SQL, inner queries or nested queries are used most frequently with the SELECT statement. The syntax of Subquery with the SELECT statement is described in the following block:

```
SELECT Column_Name1, Column_Name2,...., Column_NameN
FROM Table_Name WHERE Column_Name Comparison_Operator
( SELECT Column_Name1, Column_Name2,...., Column_NameN
FROM Table_Name WHERE condition;
```

You can create the subquery using different operation like IN,NOT IN, EXISTS,NOT EXISTS.

1. IN

EXAMPLE: Retrive the faculty name and mobile number information whose subject is 'RDBMS'

```
SQL>select fname,phno from faculty where
 fno in(select fno from subject where sname='RDBMS');
```

2. NOT IN

EXAMPLE: Retrive the faculty name and mobile number information whose subject is not 'RDBMS'

```
SQL>select fname,phno from faculty where
 fno not in(select fno from subject where sname='RDBMS');
```

3. EXISTS

EXAMPLE: Retrive the faculty no who have lecture more then 3

```
SQL>select fno from faculty where
exists(select fno from subject where lecture>3);
```

4. NOT EXISTS

EXAMPLE: Retrive the faculty no who have not lecture more then 3

```
SQL>select fno from faculty where
not exists(select fno from subject where lecture>3);
```

Subquery with the INSERT statement

We can also use the subqueries and nested queries with the INSERT statement in Structured Query Language. We can insert the results of the subquery into the table of the outer query.

Syntax

INSERT INTO Table_Name SELECT * FROM Table_Name WHERE Column_Name Operator (Subquery);

Example

Suppose we have one faculty table and one faculty_backup table with similar structure that are follows:

```
create table faculty_backup(
fno number(5) primary key,
fname char(20),
fcity char(20),
phno number(10));
```

```
insert into faculty_backup values(111,'Minesh','Surat',9954776655);
insert into faculty_backup values(112,'Chetan','Bharuch',3499776655);
```

faculty :

fno	fname	fcity	phno
101	Rajesh	Surat	9988776655
102	Vaibhav	Surat	8899776655
103	Sonal	Vadodara	7788996655

faculty_backup :

fno	fname	fcity	phno
111	Minesh	Surat	9954776655
112	Chetan	Bharuch	3499776655

```
SQL>insert into faculty select * from faculty_backup
        where fno in(select fno from faculty_backup where fno=111);
```

Subquery with the UPDATE statement

The subqueries and nested queries can be used with the UPDATE statement in Structured Query Language for updating the columns of the existing table. We can easily update one or more columns using a subquery with the UPDATE statement.

Syntax of Subquery with the UPDATE statement

```
UPDATE Table_Name SET Column_Name = New_value WHERE Value OPERATOR (SELECT COLUMN_NAME FROM

TABLE_NAME WHERE Condition) ;
```

Example : Change lecture=6 who have faculty name is vibhav.

```
SQL>update subject set lecture=6 where
        fno in(select fno from faculty where fname='Vaibhav');
```

Subquery with the DELETE statement

We can easily delete one or more records from the SQL table using Subquery with the DELETE statement in Structured Query Language.

Syntax

```
DELETE FROM Table_Name WHERE Value OPERATOR (SELECT COLUMN_NAME FROM TABLE_NAME WHERE Condition) ;
```

Example : delete record from subject who have faculty name ia vaibhav.

```
SQL>delete from subject where
            fno in(select fno from faculty where fname='Vaibhav');
```