STRUCTURED QUERY LANGUAGE

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL General DataTypes

Data type	Oracle
boolean	Byte
integer	Number
float	Number
string (fixed)	Char
string (variable)	Varchar Varchar2

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Oracle Data Types

Oracle String data types

CHAR(size)	It is used to store character data within the predefined length. It can be stored up to 2000 bytes.
NCHAR(size)	It is used to store national character data within the predefined length. It can be stored up to 2000 bytes.
VARCHAR2(size)	It is used to store variable string data within the predefined length. It can be stored up to 4000 byte.
VARCHAR(SIZE)	It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size)
NVARCHAR2(size)	It is used to store Unicode string data within the predefined length. We have to must specify the size of NVARCHAR2 data type. It can be stored up to 4000 bytes.

Oracle Numeric Data Types

NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.
FLOAT(p)	It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.
BINARY_FLOAT	It is used for binary precision(32-bit). It requires 5 bytes, including length byte.
BINARY_DOUBLE	It is used for double binary precision (64-bit). It requires 9 bytes, including length byte.

Oracle Date and Time Data Types

DATE	It is used to store a valid date-time format with a fixed length. Its range varies from January 1, 4712 BC to December 31, 9999 AD.
TIMESTAMP	It is used to store the valid date in YYYY-MM-DD with time hh:mm:ss format.

Oracle Large Object Data Types (LOB Types)

BLOB	It is used to specify unstructured binary data. Its range goes up to 2^{32}-1 bytes or 4 GB.
BFILE	It is used to store binary data in an external file. Its range goes up to 2^{32}-1 bytes or 4 GB.
CLOB	It is used for single-byte character data. Its range goes up to 2^{32}-1 bytes or 4 GB.
NCLOB	It is used to specify single byte or fixed length multibyte national character set (NCHAR) data. Its range is up to 2^{32}-1 bytes or 4 GB.
RAW(size)	It is used to specify variable length raw binary data. Its range is up to 2000 bytes per row. Its maximum size must be specified.
LONG RAW	It is used to specify variable length raw binary data. Its range up to 2^{31}-1 bytes or 2 GB, per row.

SQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a table in a database.
- Tables are organized into rows and columns; and each table must have a name.

- SQL CREATE TABLE Syntax
  - CREATE TABLE *table_name*

(
column_name1 *data_type*(*size*), *column_name2* *data_type*(*size*), *column_name3* *data_type*(*size*),
....
);

- Example
- CREATE TABLE Persons
  (
PersonID int,
LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);

- Example
- CREATE TABLE query (query_id number, query_date date);

- insert into query values (11, DATE '2021-09-01');

- SELECT * FROM query;

SQL SELECT Statement

- The SELECT statement is used to select data from a database.

- SELECT * FROM *table_name*;

- SELECT *column_name*,*column_name*
  FROM *table_name*;

SQL SELECT DISTINCT Statement

- In a table, a column may contain many duplicate values; and sometimes you only want to list the different (distinct) values.
- The DISTINCT keyword can be used to return only distinct (different) values.

  - SQL SELECT DISTINCT Syntax
  - SELECT DISTINCT *column_name*,*column_name*

FROM *table_name*;

- Example
- SELECT DISTINCT City FROM Customers;

SQL WHERE Clause

- The WHERE clause is used to extract only those records that fulfill a specified criterion.

- SQL WHERE Syntax
  - SELECT *column_name*,*column_name*

FROM *table_name*
WHERE *column_name operator value*;

- Example
- SELECT * FROM Customers
  WHERE Country='Mexico';

Operators in The WHERE Clause

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL AND & OR Operators

- The AND operator displays a record if both the first condition AND the second condition are true.
- The OR operator displays a record if either the first condition OR the second condition is true.
- Example
  - SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin';
- Example
  - SELECT * FROM Customers WHERE Country='Germany' OR City='Berlin';

SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set by one or more columns.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.
- SQL ORDER BY Syntax
  - SELECT *column_name*,*column_name*

FROM *table_name*
ORDER BY *column_name*,*column_name* ASC|DESC;

- Example
- SELECT * FROM Customers
  ORDER BY Country;
- Example
- SELECT * FROM Customers
  ORDER BY Country DESC;

SQL UPDATE Statement

- The UPDATE statement is used to update existing records in a table.

- SQL UPDATE Syntax
  - UPDATE *table_name*

SET *column1=value1*,*column2=value2*,...
WHERE *some_column=some_value*;

- Example
- UPDATE Customers

SET ContactName='Alfred Schmidt', City='Hamburg' WHERE CustomerName='Alfreds Futterkiste';

-

SQL DELETE Statement

- The DELETE statement is used to delete rows in a table.

- Example
- DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';

SQL Constraints

- SQL constraints are used to specify rules for the data in a table.
- Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).
- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- **Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.**

SQL CREATE TABLE + CONSTRAINT Syntax

CREATE TABLE *table_name*

(

column_name1 *data_type*(*size*) *constraint_name*,
column_name2 *data_type*(*size*) *constraint_name*,
column_name3 *data_type*(*size*) *constraint_name*,

....

);

- In SQL, we have the following constraints:
- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK -** Ensures that the value in a column meets a specific condition
- **DEFAULT** - Specifies a default value when specified none for this column

SQL NOT NULL Constraint

- The NOT NULL constraint enforces a column to NOT accept NULL values.
- By default, a column can hold NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

Example- **SQL NOT NULL on CREATE TABLE-** The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,  FirstName varchar(255),
Address varchar(255),  City varchar(255)
)
```

SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

ALTER TABLE Persons
MODIFY Age int NOT NULL;

SQL UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table.
- The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.
- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

- The following SQL creates a UNIQUE constraint on the "P_Id" column

when the "Persons" table is created:

- **SQL Server / Oracle / MS Access:**
- CREATE TABLE Persons

(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL, FirstName varchar(255),
Address varchar(255), City varchar(255)
)

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

SQL UNIQUE Constraint on ALTER TABLE

- To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
  ADD UNIQUE (P_Id)
```

- To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

To DROP a UNIQUE Constraint

- To drop a UNIQUE constraint, use the following SQL:
- **SQL Server / Oracle / MS Access:**
- ALTER TABLE Persons
  DROP CONSTRAINT uc_PersonID

SQL INSERT INTO Statement

- The INSERT INTO statement is used to insert new records in a table.

- INSERT INTO *table_name*
VALUES (*value1*,*value2*,*value3*,...);

- INSERT INTO *table_name* (*column1*,*column2*,*column3*,...) VALUES (*value1*,*value2*,*value3*,...);

- Example
- INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot contain NULL values.
- Most tables should have a primary key, and each table can have only ONE primary key.

- CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255), Address
  varchar(255), City varchar(255)
)

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    P_ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

SQL PRIMARY KEY Constraint on ALTER TABLE

- ALTER TABLE Persons
  ADD PRIMARY KEY (P_Id)

To DROP a PRIMARY KEY Constraint

- ALTER TABLE Persons
  DROP CONSTRAINT pk_PersonID

Difference between Primary Key and Unique Key

Primary Key	Unique Key
Unique identifier for rows of a table	Unique identifier for rows of a table when primary key is not present
Cannot be NULL	Can be NULL
Only one primary key can be present in a table	Multiple Unique Keys can be present in a table
present in a table	present in a table
Selection using primary key creates clustered index	Selection using unique key creates non-clustered index

SQL FOREIGN KEY Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
-

Person Table

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	Order No	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

```
CREATE TABLE Orders  (
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES
Persons(P_Id)
)
```

```
CREATE TABLE COURSE22(
COURSE_ID NUMBER(10) PRIMARY KEY,
COURSE_NAME VARCHAR2(50) NOT NULL,
ROLL_NO NUMBER(5),
FOREIGN KEY (ROLL_NO) REFERENCES
STUDENT22(ROLL_NO));
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

SQL FOREIGN KEY on ALTER TABLE
To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

DROP a FOREIGN KEY Constraint

SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

SQL CHECK Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

- CREATE TABLE Persons (
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL, FirstName varchar(255),
Address varchar(255), City varchar(255)
)

SQL DEFAULT Constraint

- The DEFAULT constraint is used to insert a default value into a column.

- CREATE TABLE Persons (
P_Id int NOT NULL,
LastName varchar(255) NOT NULL, FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
)

SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- SQL LIKE Syntax
- SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* LIKE *pattern*;

- SELECT * FROM Customers WHERE City LIKE 's%';

SQL Wildcards

- In SQL, wildcard characters are used with the SQL LIKE operator.
- SQL wildcards are used to search for data within a table.

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for a single character

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

The IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

- SQL IN Syntax
- SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1*,*value2*,...);

- Example
- SELECT * FROM Customers WHERE City IN ('Paris','London');

SQL BETWEEN Operator

- The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

- SQL BETWEEN Syntax
  - SELECT *column_name(s)*
    FROM *table_name*
    WHERE *column_name* BETWEEN *value1* AND *value2;*

- Example
- SELECT * FROM Products
  WHERE Price BETWEEN 10 AND 20;
- Example
- SELECT * FROM Products
  WHERE Price NOT BETWEEN 10 AND 20;

SQL Aliases

- SQL aliases are used to give a database table, or a column in a table, a temporary name.
- Basically aliases are created to make column names more readable.

- SQL Alias Syntax for Columns
- SELECT *column_name* AS *alias_name*
  FROM *table_name;*

- Example
- SELECT CustomerName AS Customer, ContactName AS [Contact Person]

FROM Customers;

SQL Alias for Tables

- Example
- SELECT o.OrderID, o.OrderDate, c.CustomerName

FROM Customers AS c, Orders AS o
WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;

SQLALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
-

  ALTER TABLE table_name
  ADD column_name datatype

- ALTER TABLE table_name
  DROP COLUMN column_name

- Now we want to add a column named "DateOfBirth" in the "Persons" table.
- ALTER TABLE Persons ADD DateOfBirth date
- Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.
- ALTER TABLE Persons
MODIFY DateOfBirth varchar(255);

- Next, we want to delete the column named "DateOfBirth" in the "Persons" table.
- ALTER TABLE Persons
DROP COLUMN DateOfBirth

The DROP TABLE Statement

- The DROP TABLE statement is used to delete a table.
  - DROP TABLE table_name

SQL Views

- A view is a virtual table.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

- SQL CREATE VIEW Syntax
  - CREATE VIEW view_name AS
    SELECT column_name(s)
    FROM table_name

WHERE condition

- Select * from view name

SQL Functions- Aggregate Functions

- SQL aggregate functions return a single value, calculated from values in a column.
- Useful aggregate functions:
- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

- SELECT AVG(Price) AS PriceAverage FROM Products;
- SELECT COUNT(CustomerID) AS OrdersFromCustomerID7 FROM Orders

WHERE CustomerID=7;

- SELECT MAX(Price) AS HighestPrice FROM Products;

SQL GROUP BY Statement

- The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

- SQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

SQL GROUP BY Examples

- The following SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

- The following SQL statement lists the number of customers in each country, sorted high to low:

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

- select city,count(personid) from persons where personid<4 group by city ;

SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

- SQL HAVING Syntax
  - SELECT column_name, aggregate_function(column_name)
    FROM table_name

WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;

SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

- Example
- select count(personid) ,city from persons group by city having count(personid) >1;
-

- Round function

- Example
- SELECT ProductName, ROUND(Price,0) AS RoundedPrice
  FROM Products;

SQL Joins

- An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

- Example
- SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders

INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

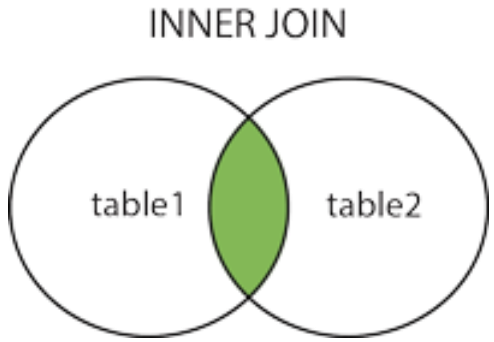
Different SQL JOINs

- **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables
- **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table
- **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table
- **FULL JOIN**: Return all rows when there is a match in ONE of the tables
-

SQL INNER JOIN Keyword

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

SQL INNER JOIN Syntax
SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name=table2.column_name*;
or:
SELECT *column_name(s)*
FROM *table1*
JOIN *table2*
ON *table1.column_name=table2.column_name*;



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

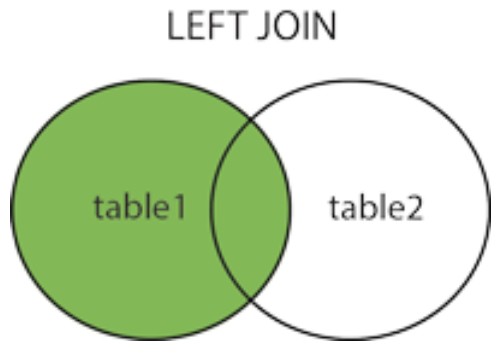
SQL INNER JOIN Example

- Example
- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;

SQL LEFT JOIN Keyword

- The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

SQL LEFT JOIN Syntax
SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name=table2.column_name*;
or:
SELECT *column_name(s)*
FROM *table1*
LEFT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;



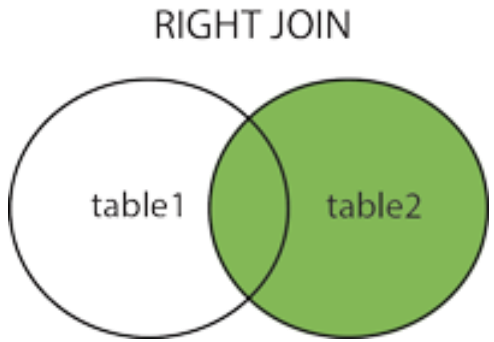
SQL LEFT JOIN Example

- Example
- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;

SQL RIGHT JOIN Keyword

- The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

SQL RIGHT JOIN Syntax
SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name=table2.column_name*;
or:
SELECT *column_name(s)*
FROM *table1*
RIGHT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;



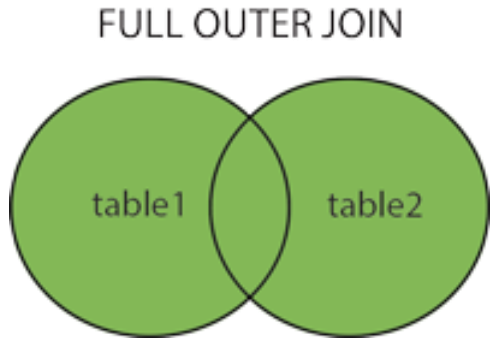
SQL RIGHT JOIN Example

- Example
- SELECT Orders.OrderID, Employees.FirstName
  FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
  ORDER BY Orders.OrderID;

SQL FULL OUTER JOIN Keyword

- The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).
- The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax
SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;



SQL FULL OUTER JOIN Example

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders
  ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;

SQL SUBQUERIES

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.
Important rules for Subqueries:

1. You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
2. A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.
3. The subquery generally executes first when the subquery doesn't have any co-relation with the main query, when there is a co-relation the parser takes the decision on the fly on which query to execute on precedence and uses the output of the subquery accordingly.
4. Subquery must be enclosed in parentheses.
5. Subqueries are on the right side of the comparison operator.
6. ORDER BY command cannot be used in a Subquery. GROUPBY command can be used to perform same function as ORDER BY command.
7. Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

Syntax: There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown below:

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
    ( SELECT COLUMN_NAME  from TABLE_NAME   WHERE ... );
```

Sample Table:

STUDENT_INFO TABLE

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988775566
Raj	102	Coimbatore	8877665544
Sasi	103	Madurai	7766553344
Ravi	104	Salem	8989898989
Sumathi	105	Kanchipuram	8989856868

STUDENT TABLE

NAME	ROLL_NO	SECTION
Ravi	104	A
Sumathi	105	B
Raj	102	A

QUESTION- To display NAME, LOCATION, PHONE_NUMBER of the students from STUDENT_INFO table whose section is A

Select NAME, LOCATION, PHONE_NUMBER from DATABASE
WHERE ROLL_NO IN
(SELECT ROLL_NO from STUDENT where SECTION='A');

Explanation : First subquery executes " SELECT ROLL_NO from STUDENT where SECTION='A' " returns ROLL_NO from STUDENT table whose SECTION is 'A'.Then outer-query executes it and return the NAME, LOCATION, PHONE_NUMBER from the DATABASE table of the student whose ROLL_NO is returned from inner subquery. Output:

INSERT INTO

Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989

Table2: Student2

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

TO INSERT STUDENT2 INTO STUDENT1 TABLE:

INSERT INTO Student1 SELECT * FROM STUDENT2;

- Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

SQL INSERT INTO SELECT Statement

- With SQL, you can copy information from one table into another.
- The INSERT INTO SELECT statement copies data from one table and inserts it into an existing table

SQL INSERT INTO SELECT Syntax
We can copy all columns from one table to another, existing table:
INSERT INTO *table2*
SELECT * FROM *table1;*
Or we can copy only the columns we want to into another, existing table:
INSERT INTO *table2 (column_name(s))* SELECT *column_name(s)* FROM *table1;*

SQL INSERT INTO SELECT Examples

Example
INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers;

Example
INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers
WHERE Country='Germany';

- To delete students from Student2 table whose rollno is same as that in Student1 table and having location as chennai

```
DELETE FROM Student2
WHERE ROLL_NO IN ( SELECT ROLL_NO
                   FROM Student1
                   WHERE LOCATION = 'chennai');
```

- Output:

```
1 row delete successfully.
```

- **Display Student2 table:**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To update name of the students to geeks in Student2 table whose location is same as Raju,Ravi in Student1 table

```
UPDATE Student2
SET NAME='geeks'
WHERE LOCATION IN ( SELECT LOCATION
                    FROM Student1
                    WHERE NAME IN ('Raju','Ravi'));
```

- Output:

```
1 row updated successfully.
```

- **Display Student2 table:**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
geeks	113	coimbatore	7878787878

ROWID

- For each row in the database, the ROWID pseudo column returns the address of the row.

- SELECT ROWID, last_name FROM employees WHERE department_id = 20;

ROWNUM

- For each row returned by a query, the ROWNUM pseudo column returns a number indicating the order in which Oracle selects the row from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on
- SELECT * FROM employees WHERE ROWNUM < 10;