SQL Concepts & Fundamentals – Sivagami S

SQL Fundamentals

Agenda

1. Introduction to SQL
2. DML,DDL,DCL
3. CRUD Operations
4. Sorting and Filtering Data
5. Summarizing Data
6. Grouping Data
7. Using Subqueries
8. JOINS
9. Views
10. Procedures

Introduction to SQL

- SQL stands for *Structured Query Language*.
- SQL lets you access and manipulate databases.
- SQL is an ANSI (American National Standards Institute) standard.

Note: Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of the SQL language.

Many standards out there:

- ANSI SQL
- SQL92 (a.k.a. SQL2)
- SQL99 (a.k.a. SQL3)

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*

General Data type

Data type	Description
CHAR(n)	Character string. Fixed-length n
VARCHAR(n)	Character string. Variable length. Maximum length n
BOOLEAN	Stores TRUE or FALSE values
INTEGER(p)	Integer numeric (no decimal). Precision p
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
ARRAY	A set-length and ordered collection of elements
XML	Stores XML data

Data types might have different names in different database. And even if the name is the same, the size and other details may be different! Always check the documentation!

Data Definition Language (DDL)

Data Definition Language (DDL) -
DDL are used to create/Alter/Drop database objects.

Examples of DDL commands:

CREATE DATABASE - Creates a new database - `CREATE DATABASE testDB;`

CREATE TABLE - Creates a new table -

```
CREATE TABLE Persons (
    PersonID int UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Address varchar(255),
    City varchar(255), PRIMARY KEY (PersonID) );
```

Data Definition Language (DDL)

ALTER TABLE- Modifies the table

```
ALTER TABLE Customers ADD Email varchar(255);
```

```
ALTER TABLE Persons MODIFY COLUMN DateOfBirth year;
```

```
ALTER TABLE Customers DROP COLUMN Email;
```

DROP TABLE - Drops (deletes) a table - `DROP TABLE Shippers;`

SQL Constraints

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. The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value
UNIQUE - Ensures that all values in a column are different
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
FOREIGN KEY - Uniquely identifies a row/record in another table
CHECK - Ensures that all values in a column satisfies a specific condition
DEFAULT - Sets a default value for a column when no value is specified
AUTO INCREMENT - used to increment values automatically

SQL Constraints

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

CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL,
      FirstName varchar(255),Age int,CHECK (Age>=18));

CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL,
       FirstName varchar(255),Age int,City varchar(255) DEFAULT 'Sandnes');

CREATE TABLE Persons (Personid int NOT NULL AUTO_INCREMENT,
      LastName varchar(255) NOT NULL,FirstName varchar(255),Age int,
      PRIMARY KEY (Personid));
```

DML Statements

SQL is divided into two main categories of statements.

1) Data Manipulation Language (DML)
DML enables you to work with the data that goes into the database.

Following commands comes under DML statements-

SELECT- Retrieves data from the database
INSERT- Inserts new data into the database
UPDATE- Updates existing data in the database
DELETE- Deletes existing data from the database

INSERT and UPDATE

INSERT

Used to enter data into table

Syntax:

```
INSERT INTO columnname
VALUES (value1, value2, ... , valuen);
```

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

UPDATE

Modify data in a table

Syntax:

```
UPDATE tablename
SET columnname = expression [, columname
= expression]
[WHERE conditionlist];
```

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City=
'Frankfurt'
WHERE CustomerID = 1;
```

ROLLBACK

Used to restore database to its previous condition.
Only applicable if COMMIT command has not been used to permanently store changes in database.

Syntax:

```
ROLLBACK;
```

COMMIT and ROLLBACK only work with data manipulation commands that are used to add, modify, or delete table rows

DELETE

DELETE

Deletes a table row

Syntax:

```
DELETE FROM tablename
[WHERE conditionlist ];
```

WHERE condition is optional

If WHERE condition is not specified, all rows from specified table will be deleted

```
DELETE FROM Customers WHERE
CustomerName='Alfreds Futterkiste';
```

SELECT Statement

DML Statements -

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

Syntax -

```
SELECT  column1,column2, ...
FROM table_name ;
```

Example -

```
SELECT emp_id,emp_name FROM employee;
```

To select all columns -

```
SELECT * FROM employee;
```

Using column alias -

```
SELECT emp_id as ‘Id’ ,emp_name as ‘Name’ FROM employee;
```

SELECT Statement

SELECT DISTINCT - The SELECT DISTINCT statement is used to return only distinct (different) values.

Syntax -

```
SELECT DISTINCT column1,column2, ...
FROM table_name ;
```

Example -

```
SELECT DISTINCT country FROM customers;
```

Getting count of distinct values -

```
SELECT COUNT(DISTINCT country) FROM customers;
```

WHERE Clause

WHERE Clause - The WHERE clause is used to filter records.

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

Syntax -

```
SELECT column1,column2, ...
FROM table_name
WHERE condition;
```

Example

```
SELECT * FROM customers
 WHERE country= 'Mexico';
```

NOTE: Numeric fields doesn't need the value to be enclosed in quotes.

Ex.

```
SELECT * FROM customers WHERE cust_id=1;
```

Operators in WHERE Clause

The following operators can be used 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

Ex –

```
SELECT customer_name FROM customers WHERE cust_id = 1;
SELECT customer_name FROM customers WHERE cust_id <> 1;
SELECT customer_name FROM customers WHERE cust_id > 1;
SELECT * FROM Customers WHERE CustomerName LIKE 'a%';
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
SELECT * FROM Customers WHERE Country IN ('Germany', 'France','UK');
```

Operators in WHERE Clause

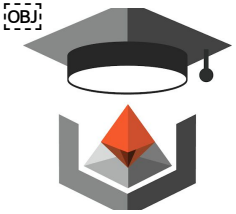
AND / OR / NOT Operator - The WHERE clause can be combined with AND, OR and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- **The AND operator displays a record if all the conditions separated by AND is TRUE.**
- **The OR operator displays a record if any of the conditions separated by OR is TRUE.**
- **The NOT operator displays a record if the condition(s) is NOT TRUE.**

Syntax - AND Operator

```
SELECT column1,column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```



Operators in WHERE Clause

Syntax - OR Operator

```
SELECT column1,column2, ...
 FROM table_name
 WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Operator

```
SELECT column1,column2, ...
 FROM table_name
 WHERE NOT condition1;
```

Example –

```
SELECT* FROM Customers WHERE Country = 'Germany' AND City= 'Berlin';
SELECT * FROM Customers WHERE City = 'Berlin' OR City = 'München';
```

Order By clause

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

```
SELECT * FROM Customers
ORDER BY Country;
```

Group By clause

The **GROUP BY** Statement. 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.

Example:

```
SELECT column1, column2
 FROM table_name
 WHERE [ conditions ]
 GROUP BY column1, column2
 ORDER BY column1, column2
```

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

Summarising clauses

GRADE	OPENING_AMT	RECEIVE_AMT	PAYMENT_AMT	OUTSTANDING_AMT	PHONE_NO	AGENT_CODE
2	6000.00	5000.00	7000.00	4000.00	BBBBBBB	A003
2	3000.00	5000.00	2000.00	6000.00	CCCCCCC	A008
3	5000.00	7000.00	6000.00	6000.00	BBBBSBB	A008
2	5000.00	7000.00	4000.00	8000.00	AVAVAVA	A011
2	4000.00	9000.00	7000.00	6000.00	FSDDSDF	A006
1	6000.00	8000.00	3000.00	11000.00	GFSGERS	A003
3	5000.00	7000.00	9000.00	3000.00	DDNRDRH	A008

```
SELECT cust_city, cust_country, MAX(outstanding_amt) FROM customer GROUP BY
cust_country, cust_city ORDER BY cust_city;
```

Alias

You can *rename a table or a column temporarily* by giving another name known as **Alias**. The use of table aliases is to rename a table in a specific SQL statement. The renaming is a temporary change and the actual table name does not change in the database. The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

The basic syntax of a **table** alias is as follows.

```
SELECT column1, column2....
FROM table_name AS alias_name
WHERE [condition];
```

The basic syntax of a **column** alias is as follows.

```
SELECT column_name AS alias_name
 FROM table_name
 WHERE [condition];
```

Subqueries

A subquery is a SQL query nested inside a larger query.

A subquery may occur in :

- A SELECT clause
- A FROM clause
- A WHERE clause

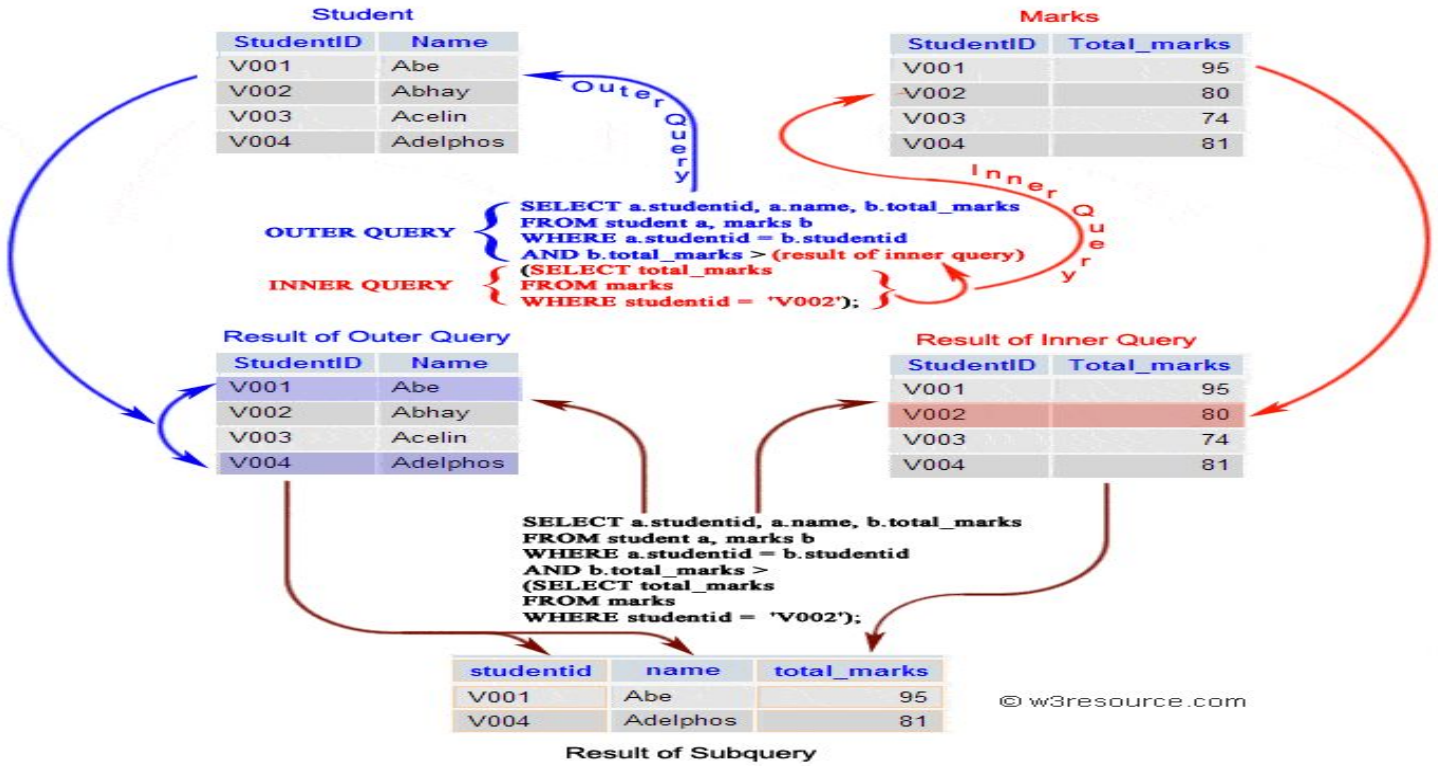
SYNTAX

```
SELECT select_list FROM table WHERE EXPR_OPERATOR
    (SELECT select_list FROM table WHERE CONDITION)
```

The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.
You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:

- Compare an expression to the result of the query.
- Determine if an expression is included in the results of the query.
- Check whether the query selects any rows.

Subqueries


Student
StudentID Name
V001 Abe
V002 Abhay
V003 Acelin
V004 Adelphos
Outer Query
Marks
StudentID Total_marks
V001 95
V002 80
V003 74
V004 81
Inner Query
OUTER QUERY
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks > (result of inner query)
INNER QUERY
(SELECT total_marks
FROM marks
WHERE studentid = 'V002');
Result of Outer Query
StudentID Name
V001 Abe
V002 Abhay
V003 Acelin
V004 Adelphos
Result of Inner Query
StudentID Total_marks
V001 95
V002 80
V003 74
V004 81
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks >
(SELECT total_marks
FROM marks
WHERE studentid = 'V002');
studentid name total_marks
V001 Abe 95
V004 Adelphos 81
Result of Subquery
© w3resource.com

Recap

TABLE 7.1 SQL Data Definition Commands

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Constraint used to validate data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and thus its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

Recap

TABLE 7.2 SQL Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values

Recap

TABLE 7.2 SQL Data Manipulation Commands (continued)

COMMAND OR OPTION	DESCRIPTION
COMPARISON OPERATORS	
=, <, >, <=, >=, <>	Used in conditional expressions
LOGICAL OPERATORS	
AND/OR/NOT	Used in conditional expressions
SPECIAL OPERATORS	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
AGGREGATE FUNCTIONS	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

Recap

TABLE 7.4 Some Common SQL Data Types

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D)	The declaration NUMBER(7,2) indicates numbers that will be stored with two decimal places and may be up to six digits long, including the sign and the decimal place. Examples: 12.32, -134.99.
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER, but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as "Smith" and "Katzenjammer" are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) will let you store characters up to 25 characters long. However, VARCHAR will not leave unused spaces. Oracle users may use VARCHAR2 as well as VARCHAR.
Date	DATE	Stores dates in the Julian date format.

JOINS in SQL

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.

Types of JOIN you can use, and the differences between them.

JOIN: Return rows when there is at least one match in both tables

LEFT JOIN: Return all rows from the left table, even if there are no matches in the right table

RIGHT JOIN: Return all rows from the right table, even if there are no matches in the left table

FULL JOIN: Return rows when there is a match in one of the tables

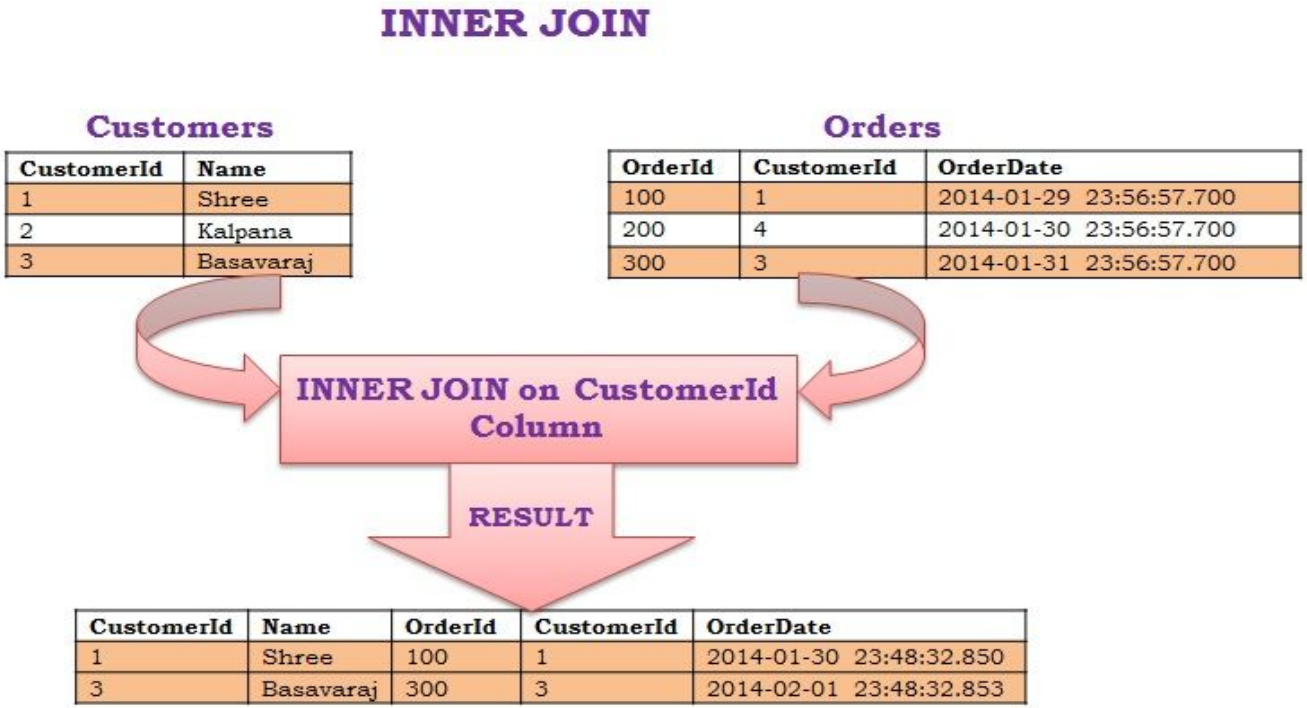
INNER JOIN

SQL INNER JOIN

Syntax:

```
SELECT
column_name(s)
FROM table_name1
INNER JOIN
table_name2
ON
table_name1.colum
n_name=table_name
2.column_name
```

INNER JOIN is the same as JOIN



LEFT OUTER JOIN

Left OuterJoin returns all the records of the left table if there are no matching records from the right table.

Syntax:

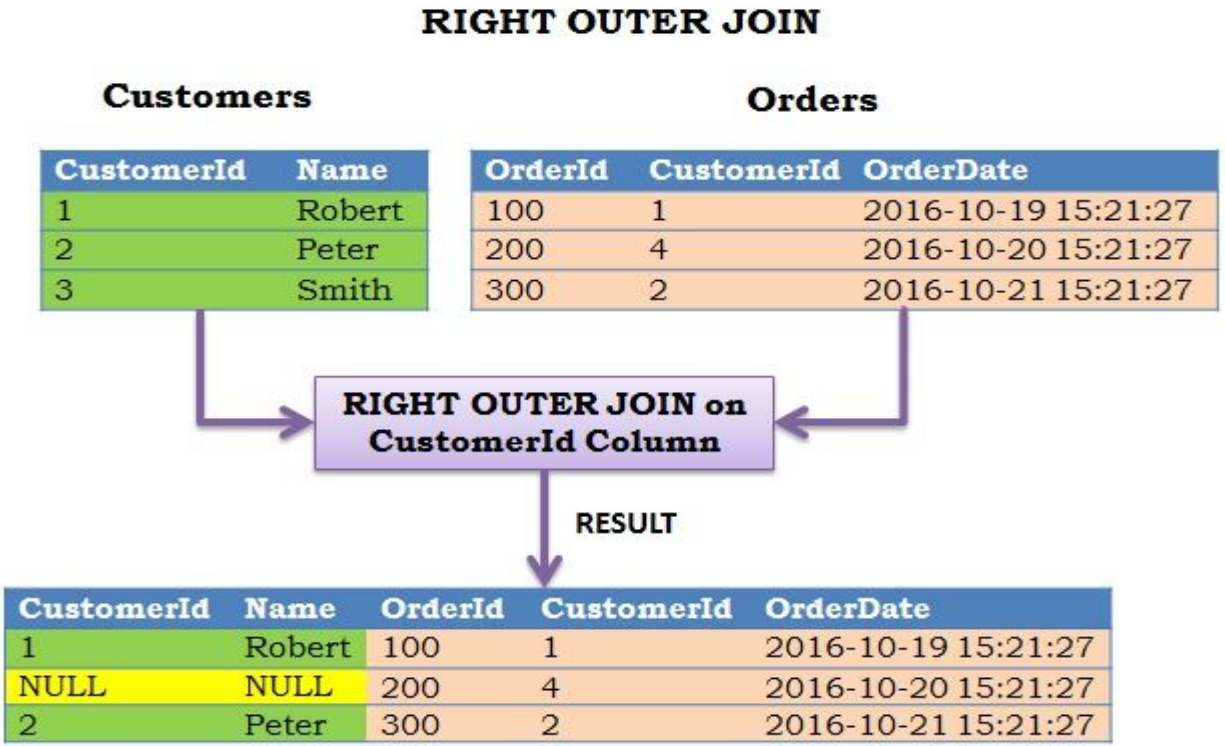
```
SELECT Column_Names(n)
FROM Table1
LEFT JOIN Table2
On table1.column1 =
table2.column1
```

RIGHT OUTER JOIN

Right OuterJoin returns all the records of the Right table if there are no matching records from the Left table.

Syntax:

```
SELECT Column_Names(n)
FROM Table1
RIGHT JOIN Table2
On table1.column1 =
table2.column1
```



FULL OUTER JOIN

FULL Outer Join return rows when there is a match in one of the tables.

Syntax:

```
SELECT Column_Names(n)
FROM Table1
FULL JOIN Table2
On table1.column1 =
table2.column1
```

Table Name: ENGLISH	
English_ID	English_Text
1	One
2	Two
3	Three
4	Four
5	Five
6	Six

Table Name: FRENCH	
French_ID	French_Text
1	Un
3	Trois
4	Quatre
5	Cinque
6	Six
7	Sept
8	Huit

RESULT SET: English FULL OUTER JOIN French				
ID	English_ID	English_Text	French_ID	French_Text
1	1	One	1	Un
2	3	Three	3	Trois
3	4	Four	4	Quatre
4	5	Five	5	Cinque
5	6	Six	6	Six
6	2	Two	NULL	NULL
7	NULL	NULL	7	Sept
8	NULL	NULL	8	Huit

Views

In SQL, a view is a virtual table based on the result-set of an SQL statement.

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.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Views

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
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	05022	Mexico
14	Chop-suey Chinese	Yang Wang	Hauptstr. 29	Bern	3012	Switzerland
15	Comércio Mineiro	Pedro Afonso	Av. dos Lusíadas, 23	São Paulo	05432-043	Brazil
16	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers WHERE Country = 'Brazil';

SELECT * FROM [Brazil Customers];
```

CustomerName	ContactName
Comércio Mineiro	Pedro Afonso
Familia Arquibaldo	Aria Cruz
Gourmet Lanchonetes	André Fonseca
Hanari Carnes	Mario Pontes
Que Delícia	Bernardo Batista
Queen Cozinha	Lúcia Carvalho
Ricardo Adocicados	Janete Limeira
Tradição Hipermercados	Anabela Domingues
Wellington Importadora	Paula Parente

Stored Procedure

A procedure is a module performing one or more actions; it does not need to return any values.

The syntax for creating a procedure is as follows:

```
CREATE OR REPLACE PROCEDURE name
[(parameter[, parameter, ...])]
AS
[local Variable declarations]
BEGIN
     executable statements
[EXCEPTION
     exception handlers]
END [name];
```

- A procedure may have 0 to many parameters.
- Every procedure has two parts:
- Parameters are the means to pass values to and from the calling environment to the server.
- These are the values that will be processed or returned via the execution of the procedure.
- There are three types of parameters: IN, OUT, and IN OUT

Stored Procedure

```
Create Procedure Procedure-name
 (
 Input parameters ,
 Output Parameters (If required)
 )
 As
 Begin
      Sql statement used in the stored procedure
 End
-----------------------------------------------------------------
/*Getstudentname is the name of the stored procedure*/

 Create  PROCEDURE Getstudentname(
 @studentid INT                    --Input parameter ,  Studentid of the student
 )
 AS
 BEGIN
 SELECT Firstname+' '+Lastname FROM tbl_Students WHERE studentid=@studentid
 END
```

Stored Procedure

With OUTPUT Parameter

```
Create  PROCEDURE GetstudentnameInOutputVariable
 (
 @studentid INT,                        --Input parameter ,  Studentid of the student
 @studentname VARCHAR(200)  OUT         -- Out parameter declared with the help of OUT keyword
 )
 AS
 BEGIN
 SELECT @studentname= Firstname+' '+Lastname FROM tbl_Students WHERE studentid=@studentid
 END
```

EXECUTION OF Stored Procedure

```
Execute Getstudentname 1
Exec Getstudentname 1

Declare @Studentname as nvarchar(200)   -- Declaring the variable to collect the Studentname
Declare @Studentemail as nvarchar(50)     -- Declaring the variable to collect the Studentemail
Execute GetstudentnameInOutputVariable 1 , @Studentname output, @Studentemail output
select @Studentname,@Studentemail      -- "Select" Statement is used to show the output from Procedure
```

Any Question?

Do it yourself

01 Exercise 1

Here is the assignment should be done by the time.

02 Exercise 2

Here is the assignment should be done by the time.

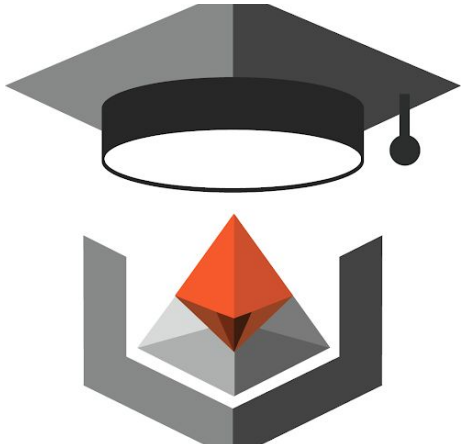
03 Exercise 3

Here is the assignment should be done by the time.

au.communication@accoliteindia.com

www.accolite.com

Thank You



twitter.com/weareaccolite

facebook.com/accolite

linkedin.com/company/accolite