WORKING WITH DATABASES & TABLES

WHAT IS SQL

SQL stands for –Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for database management system.

SQL statements are used to perform tasks such as update on a database, or retrieve data from a database. The standard SQL commands such as Select, Insert, Update, Delete, Create, and Drop can be used to accomplish almost everything that one needs to do with a database.

Query is statement requesting retrieval of information. In short, SQL is

- Stands for–Structured Query Language
- Allows you to access databases.
- Is an ANSI (American National Standards Institute) standard computer language.
- Can execute queries against a database.
- Can retrieve data from a database.
- Can insert new records in a database.
- Can delete records from a database.
- Can update records in a database.

SQL is a standard language that works with database programs like MS Access, DB2, MS SQL Server, Oracle, MySQL, and other database systems, although most of them also have their own additional proprietary extensions that are usually only used on their system.

BASIC STRUCTURE OF SQL

The basic structure of an SQL expression consists of three clauses:

- ✓ Select
- ✓ From
- ✓ Where

- The **SELECT** statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

- The **FROM** command is used to specify which table to select or delete data from.

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

SQL Database Table

When creating tables, you must provide:

- Table name
- Column name(s)
- Data types for each column

Guidelines for creating tables:

Table and column naming rules

- ✓ Must start with a letter, which is followed by a sequence of letters, numbers, _, #, or $
- ✓ Must be 1 to 30 characters long
- ✓ Must not be an Oracle server reserved password

ORACLE DATATYPES

Data Type	Description	Size
Integer	It can store numeric data.	A NUMBER value requires From 1 to 22 bytes.
VARCHAR(size [BYTE | CHAR])	Variable-length character string.	From 1 byte to 4KB.
DATE	Valid date range : From January 1, 4712 BC, to December 31, 9999 AD. Date format: [DD-MON-YY]	The size is fixed at 7 bytes.
TIMESTAMP [(fractional_ seconds_precision)]	This data type contains the date time fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It contains fractional seconds but does not have a time zone.	The size is 7 or 11 bytes, Depending on the precision.
CHAR [(size [BYTE | CHAR])]	Fixed-length character data of length size bytes Or characters.	Maximum size is 2000 bytes [2 KB] or characters. Default And minimum size is 1 byte.
CLOB	A character large object containing single-byte or multibyte characters. [can store Unicode characters]	Maximum size is (4 gigabytes - 1) * (Database blocks size).
BLOB	A binary large object. [stores image, having more size capacity than clob]	Maximum size is 4 gigabytes.
NVARCHAR2(size)	Variable-length Unicode character string having maximum length size characters.	Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. **You must specify size for NVARCHAR2.**
BINARY_FLOAT	32-bit floating point number.	This data type requires

		4 bytes.
BINARY_DOUBLE	64-bit floating point number.	This data type requires 8 bytes.

SQL DDL and DML

SQL can be divided into two parts: the Data Manipulation Language (DML) and the Data Definition Language (DDL).

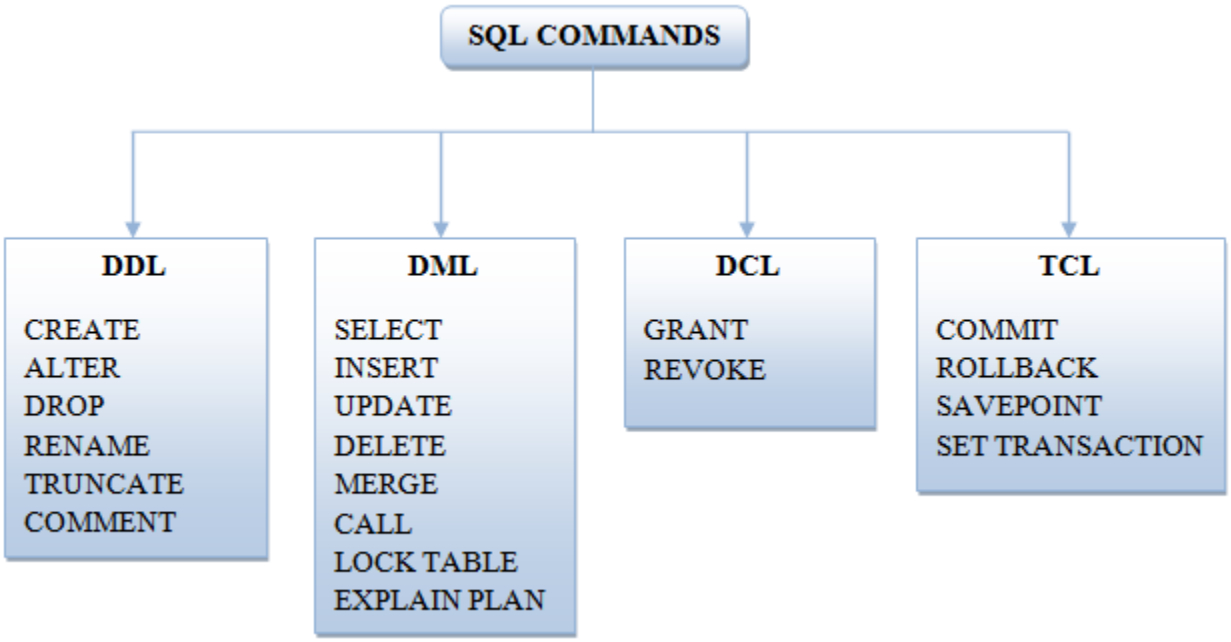
The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specify links between tables, and impose constraints between tables.

The most important DDL statements in SQL are:

- **CREATE TABLE** – creates a new table
- **ALTER TABLE** – modifies a table
- **DROP TABLE** – deletes a table
- **TRUNCATE**-deletes all records
- **RENAME**-rename old table name to new table name

The query and update commands form the DML part of SQL:

- **SELECT** – extracts data from a database
- **UPDATE** – updates data in a database
- **DELETE** – deletes data from a database
- **INSERT INTO** – inserts new data into a database


SQL COMMANDS
DDL
CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT
DML
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
LOCK TABLE
EXPLAIN PLAN
DCL
GRANT
REVOKE
TCL
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

DDL STATEMENTS

SQL CREATE TABLE STATEMENT

The SQL syntax for **CREATE TABLE** is

CREATE TABLE table_name (column1 datatype, column2 datatype, ...);

Example: -

```
CREATE TABLE Persons(
   personID number,
   lname varchar(255),
   fname varchar(255),
   address varchar(255),
   city varchar(255)
);
```

SQL DROP TABLE STATEMENT

The DROP TABLE statement is used to delete a table from the database.

Syntax:

DROP TABLE table_name;

Example:

DROP TABLE Persons;

That will delete the Persons table from the database along with its data.

SQL RENAME TABLE STATEMENT

SQL RENAME TABLE syntax is used to change the name of a table. Sometimes, we choose non-meaningful name for the table. So it is required to be changed.

Syntax:

RENAME old_table _name To new_table_name;

Example:

RENAME STUDENTS TO ARTISTS;

After that the table "students" will be changed into table name "artists" .

SQL TRUNCATE TABLE STATEMENT

The TRUNCATE TABLE command deletes the data inside a table, but not the table itself.

The following SQL truncates the table "C":

Syntax:

Truncate table table_name;

Example:
Truncate table customer;

SQL ALTER TABLE STATEMENT

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

Syntax:
To add a column in a table, use the following syntax:
ALTER TABLE table_name ADD column_name datatype;

To delete a column in a table, use the following syntax (that column must be empty):

ALTER TABLE table_name DROP COLUMN column_name;

ALTER TABLE ADD column example:

Look at the–Persons table:

P_id	Lastname	Firstname	Address	City
1	Banner	Bruce	Sonifalia	Surat
2	Stark	Tony	Navsari	Surat
3	Klein	Daniel	Nanpura	Surat

Now we want to add a column named –DateOfBirth in the

Persons table. ALTER TABLE Persons ADD DateOfBirth date;
Notice that the new column, DateOfBirth, is of type date and is going to hold a date. The data type specifies what type of data the column can hold.

The–Persons table will now like this:

P_id	Lastname	Firstname	Address	City	DateOfBirth
1	Banner	Bruce	Sonifalia	Surat	
2	Stark	Tony	Navsari	Surat	
3	Klein	Daniel	Nanpura	Surat	

Delete the column example:
Next, we want to delete the column named –DateOfBirth in the –Persons

table. ALTER TABLE Persons DROP COLUMN DateOfBirth;

That removes the column DateOfBirth from the Persons table.

DDL STATEMENTS

The SQL INSERT INTO Statement

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

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1: Specify both the column names and the values to be inserted:

Syntax:

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

Example:

```
INSERT INTO Customers(CustomerName,ContactName,Address,City,PostalCode,Coun
try)
VALUES ('Cardinal', 'Tom', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

2: If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

Syntax:

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

Values:

```
INSERT INTO Customers VALUES ('Cardinal', 'Tom', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

SQL QUERIES

Most of the actions you need to perform on a database are done with SQL statements.

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

If using MS Access so, we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

MANAGING CONSTRAINTS & RELATIONSHIPS

PRIMARY & FOREIGN KEY

Primary key is basically a constraint. If defined on any column you would not be able to insert anything which is already there in that particular column. i.e. the value of that particular column can't be duplicated.

To establish a link between two tables or in other words to have primary-Foreign key relationship between two or more tables, it is to have primary key defined on at least one of the tables.

Primary key of one table act as the foreign key in other table. A foreign key column can have duplicate values. Once you establish a primary-foreign key relationship between two or more table it's like having a constraint on all the tables in the relationship. Such constraint is called Referential Integrity' constraint. The table having primary key column is called Parent table all the other tables linked to the parent table via foreign key are called child tables.

Once this Primary-Foreign key relationship is defined in between tables, the following constraints get effective:

- You can't delete any record from the parent table if there are corresponding records in the child tables.
- You can't insert any record in the child table if there is no corresponding record in the parent table

Let's look at the example to completely understand the concept behind these constraints. Consider one table having name as supplier's and second table

Supplier_Id	Supplier_name
10000	IBM
10001	IBM
10002	MICROSOFT
10003	MICROSOFT

Order_Id	Supplier_Id	Order_Date
500125	10000	2-2-2002
500126	10001	2-3-2003

having the name as orders with the following data in the corresponding:
Once you have defined a primary-foreign key relationship between these two tables with supplier_id of_suppliers' table as primary key and supplier_id of_orders' table as foreign key, if you try to delete supplier_id record 10001 from supplier table, Oracle will not let you do so. To delete such type of records you have to first delete all the corresponding records from the child tables. Only then you would be able to delete the record from the parent table.

HOW TO CREATE A PRIMARY KEY?

The syntax to create a primary key using the CREATE TABLE statement in Oracle/PLSQL is:

```
CREATE TABLE table_name
(
  Column1 datatype null/not null Primary key,
  Column2 datatype null/not null,
);
```

Ex: - create table suppliers (s_id number primary key, s_name varchar(30));

HOW TO CREATE FOREIGN KEY (RELATIONSHIP)?

Primary key of parent table is the foreign key of the child table. This you should remember always. Now let's see the step by step process on defining the primary-foreign key relationship in between tables. You are allowed to enter duplicate values in the foreign key column.

The syntax to create a foreign key using the CREATE TABLE statement in Oracle/PLSQL is:

```
CREATE TABLE table_name
(
  Column1 datatype null/not null Primary key,
  Column2 datatype null/not null,
  Column2 constraint constraint_name references parent_table (column1)
);
```

Ex: -create table orders (o_id number primary key, s_id constraint s_id_fk references suppliers (s_id), o_date date);

Suppose you have selected two tables; Dept and Emp. In Dept we have Dno as primary key. To develop primary-foreign key relationship,

Over here if you check the Cascade Delete Related Records', you would be able to delete records from the parent table even if there are corresponding records in the child table. When you check this option, Oracle automatically first delete all the related records from the child tables and then delete the parent records.

On the other hand if you select the Cascade Update Related Fields', any update operation on the primary key column will be having a ripple effect on the child table. In other words all the corresponding child tables' foreign key column values will get updated. Finally click the ok button once done.

Duplicates are allowed in foreign key column whereas like explained before duplicates are not allowed in primary key column.

The SQL INSERT INTO STATEMENT

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

Syntax:

The INSERT INTO statement is used to insert the records/ data in tables using SQL query. It is possible to write the INSERT INTO statement in two forms:

a) The first form doesn't specify the column names where the data will be inserted, only their values:
INSERT INTO table_name VALUES (value1, value2...);

b) The second form specifies both the column names and the values to be inserted: INSERT INTO table_name (column1, column2...) VALUES (value1,value2...);

Example:

We have the following–customer table:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat

Now we want to insert a new row in the –customer table. We use the following SQL statement:

Example: INSERT INTO customer VALUES (106,'Peter','Banner','Pune');

The –customer table will now look like this:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat
106	Peter	Banner	Pune

- **Insert data only in specified columns:**

The following SQL statement will add a new row, but only add data in the –cust_id , –Firstname columns:

Example: INSERT INTO customer (cust_id, Firstname) VALUES (107,'Vinay'); The –customer

table will look like this:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat
106	Niranjana	Banner	Pune
107	Vinay		

SQL UPDATE STATEMENT

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

Syntax:

UPDATE table_name SET column1=value, column2=value2... WHERE some_column= some_value;

Example: table: - customer

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat
106	Niranjana	Banner	Pune

Now we want to update Lastname of–Robert as—Banner instead of–Patrik.

We have:

Example: UPDATE customer SET Lastname='Banner' WHERE Firstname='Robert';

Update Multiple Columns

Syntax:

UPDATE table_name SET column1=value, column2=value2... WHERE some_column= some_value;

Example: table: - customer

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat
106	Niranjana	Banner	Pune

Example: UPDATE customer SET Firstname='Deny', Lastname='Banner' where cust_id=105;
Output:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Banner	Bhopal
105	Deny	Banner	Surat
106	Niranjana	Banner	Pune

SQL DELETE STATEMENT

The DELETE statement is used to delete existing record in a table.

Syntax:
DELETE FROM table_name WHERE column=value;

Example: table: - customer

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat
106	Niranjana	Banner	Pune

Now we want delete record of firstname='Daniel'. We have:
Example: DELETE FROM customer WHERE Firstname='Daniel';

OR

DELETE FROM customer WHERE cust_id=103;

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat
106	Niranjana	Banner	Pune

It will delete record of –Daniel.

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

Syntax:

```
DELETE FROM table_name;
```

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

```
DELETE FROM Customers;
```

THE SQL SELECT STATEMENT

The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set.

Syntax:

SELECT col1,col2,… FROM table_name;

NOTE: SQL is not case sensitive. SELECT is the same as select.

EXAMPLE:

To select the content of columns named –lastname and –firstname, from the database table called customer, use a SELECT statement like this:
The database table–customer has the following contents:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal

The results from the SELECT statement are:
SELECT firstname, lastname FROM customer;

Firstname	Lastname
Bruce	Banner
Tony	Stark
Daniel	Klein
Robert	Patrik

1) SELECT *

To select all columns from the table, use * symbol instead of column names.

Syntax:
SELECT * FROM table_name;

Like this:
SELECT * FROM customer;

Display all the records of the table customer.

2) The SELECT DISTINCT statement

In a table, some of the columns may contain duplicate values. This is not a problem; however, sometimes you will want to list only the different values in a table.

The DISTINCT keyword can be used to list only different (distinct) values in SELECT statement.

Syntax:
SELECT DISTINCT column_name(s) FROM table_name;

Example:
The customer table:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal

Now we want to select only the distinct values from the column named –City from the table above. We use the following SELECT statement:

Example: SELECT city FROM customer;

City
Surat
Surat
Surat
Bhopal

But if we use DISTINCT word then:

Example: SELECT DISTINCT city FROM customer;

The result-set will look like this:

City
Bhopal
Surat

- –Surat is listed only once.

3) The SQL WHERE clause

The WHERE clause is used to specify a selection criterion.

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

Syntax:
SELECT column_name(s) FROM table_name WHERE column_name operator value;

The following operators can be used with the WHERE clause:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
Between	Between an inclusive range
Like	Search for a pattern(%, _)
IN	If you know the exact value you want to return for atleast one of the columns

Example:
The customer table:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal

Now we want to select only the persons living in the city –Surat from the table above. We use the following SELECT statement:

Example: SELECT * FROM customer WHERE city='Surat';

The result will look like this:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat

102	Tony	Stark	Surat
103	Daniel	Klein	Surat

4) SQL AND & OR Operators

The AND & OR operators are used to filter records based on more than one condition in WHERE clause.

The AND operator displays a record if both the first condition and the second condition is true. The OR operator displays a record if either the first condition or the second condition is true.

- ✓ **AND Operator Example:**

The customer table:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat

Now we want to select only the persons with the first name equal to –Robert AND the last name equal to –Patrik:

We use the following SELECT statement:
Example: SELECT * FROM customer WHERE Firstname='Robert' AND Lastname='Patrik';

The result-set will look like this:

cust_id	Firstname	Lastname	City
104	Robert	Patrik	Bhopal

- ✓ **OR Operator Example:**

Now we want to select only the persons with the firstname equal to –Robert OR the firstname equal to –Bruce:

We use the following SELECT statement:
Example: SELECT * FROM customer WHERE Firstname='Robert' OR Firstname='Bruce';

The result-set will look like this:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
104	Robert	Patrik	Bhopal

✓ **Combining AND & OR Operators:**

You can also combine AND and OR (use parenthesis to form complex expressions).

Now we want to select only the persons with the lastname equal to –Banner AND the first name equal to –Bruce OR to –Robert:

We use the following SELECT statement:
Example: SELECT * FROM customer WHERE Lastname='Banner' AND (Firstname='Bruce' OR Firstname='Robert');

The result-set will look like this:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat

5) SQL LIKE Operator

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

Syntax:
SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;

The %,_ sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Example:
The–customer table:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat

Now we want to select the customers whose first name

starts with –K We use the following SELECT statement:
Example: SELECT * FROM customer WHERE Firstname LIKE K%';

The result will look like:

cust_id	Firstname	Lastname	City
105	Kiaan	Patrik	Surat

The following SQL statement will return customers with the first names that end with an e:

Example: SELECT * FROM customers WHERE Firstname LIKE %e';

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat

The ORDER BY Keyword:

The ORDER BY keyword is used to sort the result-set.
The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sorts the records in ascending order by default.
If you want to sort the records in a descending order, you can use the DESC keyword.

Syntax:
SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC;

Example: table:- customer

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
102	Tony	Stark	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat

Now we want to select all the customers from the table above, however, we want to sort the customers by their last name.

We use the following SELECT statement:
Example: SELECT * FROM customer ORDER BY Lastname;

The result-set will look like this:

cust_id	Firstname	Lastname	City
102	Tony	Stark	Surat
105	Kiaan	Patrik	Surat
104	Robert	Patrik	Bhopal
103	Daniel	Klein	Surat
101	Bruce	Banner	Surat

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:
Example: SELECT * FROM customer ORDER BY Lastname DESC;

The result-set will look like this:

cust_id	Firstname	Lastname	City
101	Bruce	Banner	Surat
103	Daniel	Klein	Surat
104	Robert	Patrik	Bhopal
105	Kiaan	Patrik	Surat
102	Tony	Stark	Surat

Group By

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.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Example:

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

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

EXISTS / NOT EXISTS

The EXISTS operator is used to test for the existence of any record in a subquery. The EXISTS operator returns true if the subquery returns one or more records.

Ex: Select * from customers where exists (select * from order_details where customer.customer_id=order_details.customer_id);

This Oracle EXISTS condition example will return all records from the customers table where there is at least one record in the order_details table with the matching customer_id.

The Oracle EXISTS condition can also be combined with the NOT operator.

Ex: Select * from customers where not exists (select * from order_details where customer.customer_id=order_details.customer_id);

This Oracle EXISTS example will return all records from the customers table where there are no records in the order_details table for the given customer_id.

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Syntax:

```
SELECT column_name AS alias_name
FROM table_name;
```

Example:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

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;
```

SQL TABLE constraints:

Constraints are used to limit the type of data that can go into a table. Constraints can be specified when a table is created or after the table is created. Common types of constraints include the following:

- **NOT NULL constraint**: Ensures that a column can't have NULL value.
- **UNIQUE constraint:** Ensures that all values in a column are different.
- **PRIMARY KEY constraint:** used to uniquely identify a row in the table.
- **FOREIGN KEY constraint:** used to ensure referential integrity of the data.

1) SQL NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values.
The NOT NULL constraint enforces a field to always contain a value. This means that you can't insert a new record, or update a record without adding a value to this field.

Example:

CREATE TABLE empdetails (eno integer, ename varchar(10), DOB date NOT NULL);

Columns – eno and –ename can include NULL, while–DOB can't include NULL.

2) SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table.
The UNIQUE and PRIMARY KEY constraint 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.

Example:

CREATE TABLE persons (p_id number NOT NULL UNIQUE, lastname varchar (10) NOT NULL, firstname varchar (10), address varchar (30), city varchar (10));

No repetition in this constraint.

Difference between Primary Key and Unique Constraint

UNIQUE contains NULL where as PRIMARY KEY doesn't contain NULL.

Only 1 PRIMARY KEY per table where as more than 1 UNIQUE KEY can be applied.

3) SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.
A PRIMARY KEY column can't contain NULL values.

Each table should have a PRIMARY KEY, and each table can have only one PRIMARY KEY.

Example:

CREATE TABLE persons (p_id integer PRIMARY KEY, sname varchar(10) NOT NULL, address varchar(30), city varchar(10));

4) SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table. Lets illustrate the foreign key with an example. Look at the following two tables: The –Persons table:

P_id	Lastname	Firstname	Address	City
1	Banner	Bruce	Sonifalia	Surat
2	Stark	Tony	Navsari	Surat
3	Klein	Daniel	Nanpura	Surat

The–Orders table:

O_id	OrderNo	P_id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that –P_id column in the–Orders table points to the P_id column in the–Persons table. The–P_id column in the–Persons table is the PRIMARY KEY in the –Persons table.
The–P_id column in the–Orders table is a FOREIGN KEY in the–Orders table.
The FOREIGN KEY constraint is used to prevent actions that would destroy link between tables. The FOREIGN KEY constraint also prevents that invalid data is inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

Example:

CREATE TABLE Orders (O_id number PRIMARY KEY, OrderNo number NOT NULL,
P_id number REFERENCES Persons (P_id));

5) Check Constraint

A check constraint allows you to specify a condition on each row in a table.

Syntax:

```
CREATE TABLE table_name
(
  column1 datatype,
  column2 datatype,
 ...
  CONSTRAINT constraint_name CHECK (column_name condition)
);
```

Example:

```
CREATE TABLE suppliers
(
  supplier_id numeric(4),
  supplier_name varchar2(50),
  CONSTRAINT check_supplier_id
  CHECK (supplier_id BETWEEN 100 and 9999)
);
```

6) Default Constraint

The DEFAULT constraint is used to provide a default value for a column.
The default value will be added to all new records IF no other value is specified.

Example:

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

CASCADE UPDATE & DELETE

Cascading referential integrity constraints allow you to define the actions Open Office Base takes when a user attempts to delete or update a key to which existing foreign keys point.

The REFERENCES clauses of the CREATE TABLE and ALTER TABLE statements support ON DELETE and ON UPDATE clauses:

- [ON DELETE { CASCADE | NO ACTION }]
- [ON UPDATE { CASCADE | NO ACTION }]

NO ACTION is the default if ON DELETE or ON UPDATE is not specified. NO ACTION specifies the same behavior that occurs in earlier versions of SQL Server.

ON DELETE NO ACTION

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE is rolled back.

ON UPDATE NO ACTION

Specifies that if an attempt is made to update a key value in a row whose key is referenced by foreign keys in existing rows in other tables, an error is raised and the UPDATE is rolled back. CASCADE allows deletions or updates of key values to cascade through the tables defined to have foreign key relationships that can be traced back to the table on which the modification is performed. CASCADE cannot be specified for any foreign keys or primary keys that have a **timestamp** column.

ON DELETE CASCADE

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the rows deleted from those tables.

ON UPDATE CASCADE

Specifies that if an attempt is made to update a key value in a row, where the key value is referenced by foreign keys in existing rows in other tables, all of the foreign key values are also updated to the new value specified for the key. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the key values updated in those tables.

Examples of cascading referential actions can be based on the **FK_Products_Suppliers** constraint on the **Products** table in **Northwind**. This constraint establishes a foreign key relationship from the **SupplierID** column in the **Products** table to the **SupplierID** primary key column in the **Suppliers** table. If ON DELETE CASCADE is specified for the constraint, deleting the row in **Suppliers** where **SupplierID** equals 1 also deletes the three rows in **Products** where **SupplierID** equals 1. If ON UPDATE CASCADE is specified for the constraint, updating the **SupplierID** value in the **Suppliers** table from 1 through 55 also updates the **SupplierID** values in the three rows in **Products** whose **SupplierID** values currently equal 1.

Cascading actions cannot be specified for a table that has an INSTEAD OF UPDATE or INSTEAD OF DELETE trigger. After a cascading action has been defined for a table,an INSTEAD OF UPDATE or INSTEAD OF DELETE trigger cannot be added toit.

SQL FUNCTIONS

Aggregate Functions:

It is commonly used with the aggregate functions together with the GROUP BY clause. The GROUP BY clause divides the rows into groups and an aggregate function calculates and returns a single result for each group.

If you use aggregate functions without a GROUP BY clause, then the aggregate functions apply to all rows of the queried tables or views.

We also use the aggregate functions in the HAVING clause to filter groups from the output based on the results of the aggregate functions.

Oracle aggregate functions can appear in SELECT lists and ORDER BY, GROUP BY, and HAVING clauses.

1) COUNT

- The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types. All aggregate functions by default exclude nulls values before working on the data.
- COUNT (*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (*) also considers Nulls and duplicates.

reference_ number	transaction_ date	return_date	membership_ number	movie_id	movie_ returned
11	20-06-2012	NULL	1	1	0
12	22-06-2012	25-06-2012	1	2	0
13	22-06-2012	25-06-2012	3	2	0
14	21-06-2012	24-06-2012	2	2	0
15	23-06-2012	NULL	3	3	0

Let's suppose that we want to get the number of times that the movie with id 2 has been rented out

SELECT COUNT(movie_id) FROM movierentals WHERE movie_id = 2;

COUNT(movie_id)
3

DISTINCT Keyword

The DISTINCT keyword that allows us to omit duplicates from our results. This is achieved by grouping similar values together.

SELECT movie_id FROM movierentals;

movie_id
1

2
2
2
3

Now let's execute the same query with the distinct keyword –

SELECT DISTINCT movie_id FROM movierentals;

As shown below , distinct omits duplicate records from the results.

movie_id
1
2
3

2) MIN

- The MIN function returns the smallest value in the specified table field.
- As an example, let's suppose we want to know the year in which the oldest movie in our library was released, we can use MySQL's MIN function to get the desired information.

SELECT MIN(year_released) FROM movies;

MIN('year_released')
2005

3) MAX

- MAX function is the opposite of the MIN function. It returns the largest value from the specified table field.
- Let's assume we want to get the year that the latest movie in our database was released. We can easily use the MAX function to achieve that.

The following example returns the latest movie year released.
SELECT MAX(`year_released`) FROM `movies`;

MAX(year_released)
2012

4) SUM

- SUM function which returns the sum of all the values in the specified column.
- SUM works on numeric fields only. Null values are excluded from the result returned.

The following table shows the data in payments table-

payment_ id	membership_ number	payment_ date	description	amount_ paid	external_ reference _number
1	1	23-07-2012	Movie rental payment	2500	11
2	1	25-07-2012	Movie rental payment	2000	12
3	3	30-07-2012	Movie rental payment	6000	NULL

The query shown below gets the all payments made and sums them up to return a single result.

SELECT SUM(amount_paid) FROM payments;

Executing the above query in MySQL workbench against the myflixdb gives the following results.

SUM(amount_paid)
10500

5) AVG

- AVG function returns the average of the values in a specified column. Just like the SUM function, it works only on numeric data types.
- Suppose we want to find the average amount paid. We can use the following query -

SELECT AVG(amount_paid) FROM payments;

6) FIRST

- The FIRST() function returns the first raw value of the selected column.
- It works in MS Access only.
- **Syntax:** SELECT FIRST(column_name) FROM table_name;
- **Example:** SELECT FIRST(MARKS) AS MarksFirst FROM Students;

7) LAST

- The LAST() function returns the last value of the selected column.
- It works in MS Access only.
- **Syntax:** SELECT LAST(column_name) FROM table_name;
- **Example :**SELECT LAST(MARKS) AS MarksLast FROM Students;

Scalar Functions

1) **Lower or Lcase**
   - Convert the string from upper case to lower case.
   - **Syntax** : lower('string') , lower(fieldname)
   - **Example** : lower('William') = william , lower(ename) = william

2) **Upper or Ucase**
   - Convert the string from lower case to upper case.
   - **Syntax** : upper('string') , upper(fieldname)
   - **Example** : upper('william') = William

3) **Round**
   - Its return the nearest upper value or lower value depends of given floating value.
   - **Syntax** : round(float value) , round(fieldname)
   - **Example** : round(12.67) = 13 , round(12.47) = 12 , round(commission) = 670

4) Mid
   - The MID() function is used to extract characters from a text field.
   - **Syntax:** SELECT MID(column_name,start,length) AS some_name FROM table_name;
   - **Example :** SELECT MID(City,1,4) AS ShortCity
     FROM Customers;

Sequence

Sequence is a one type of oracle object which is use to generate the numeric value in manner of sequence number.
Sequence is use to generate a number in sequence manner either in order or non order.
The value generate by sequence is maximum 38 digit number

Sequence is a set of integers 1, 2, 3, ... that are generated and supported by some database systems to produce unique values on demand.

- A sequence is a user defined schema bound object that generates a sequence of numeric values.
- Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.
- The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when max_value exceeds.

Syntax:

```
CREATE SEQUENCE sequence_name
START WITH initial_value
INCREMENT BY increment_value
MINVALUE minimum value
MAXVALUE maximum value
CYCLE|NOCYCLE ;
```

sequence_name: Name of the sequence.

initial_value: starting value from where the sequence starts.
Initial_value should be greater than or equal
to minimum value and less than equal to maximum value.

increment_value: Value by which sequence will increment itself.
Increment_value can be positive or negative.

minimum_value: Minimum value of the sequence.
maximum_value: Maximum value of the sequence.

cycle: When sequence reaches its set_limit
it starts from beginning.

nocycle: An exception will be thrown
if sequence exceeds its max_value.

Example

Following is the sequence query creating sequence in ascending order.

```
CREATE SEQUENCE s_1
start with 1
increment by 1
minvalue 0
maxvalue 100
cycle;
```

Above query will create a sequence named *s_1*.Sequence will start from 1 and will be incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.

Following is the sequence query creating sequence in descending order.

```
CREATE SEQUENCE s_2
start with 100
increment by -1
minvalue 1
maxvalue 100
cycle;
```

Above query will create a sequence named *s_2*.Sequence will start from 100 and should be less than or equal to maximum value and will be incremented by -1 having minimum value 1.

Example to use sequence: create a table named students with columns as id and name.

```
CREATE TABLE students
(
ID number(10),
NAME char(20)
);
```

Now insert values into table

```
INSERT into students VALUES(s_1.nextval,'Ramesh');

INSERT into students VALUES(s_1.nextval,'Suresh');
```

where *s_1.nextval* will insert id's in id column in a sequence as defined in sequence_1.

Output:

```
 ______________________
| ID  |      NAME      |
------------------------
|  1  |     Ramesh     |
|  2  |     Suresh     |
 ----------------------
```

View

After a table is created and population with data, it may become necessary to prevent all user from accessing all column of a table for data security reason.

This would mean creating several tables having the appropriate number of column and accessing specific user to each table as required.

This will answer data security requirement very well but give rise to a great deal of redundant data being resident in table in the database.

To reduce redundant data to minimum possible. Oracle allow the creation of an object called view. Some view are use only for looking at the table data and other view can be use to insert, update and delete table data as well as view data if the view use to only look at table data and nothing else the view is called read only view.

A view that is used to look at table data as well as insert, update and delete table data is called updatable view.

Reason when view are created.

- When data security is require.
- When data redundancy is to kept to the minimum while maintaining data security.

Syntax :

CREATE VIEW <viewname> AS
SELECT <columns> FROM <tablename> WHERE <condition>;

Example :

CREATE VIEW v1 AS
SELECT * FROM emp;

CREATE VIEW v1 AS
SELECT eno,ename FROM emp;

Renaming view name

The column of the view can take on different names from the table column.

Syntax :
CREATE VIEW <viewname> AS
SELECT <column1> "new_name",<column2> "new_name" FROM <tablename>
WHERE <condition>;

Example :
CREATE VIEW v3 AS
SELECT eno "employee" FROM emp;

Selecting data from view Syntax :
SELECT <columns> FROM <viewname> WHERE <condition>;

Example :
SELECT * FROM v1;

Updatable view

View can also be used for data manipulation. View on which data manipulation can be done are called updatable view.
When updatable view name is given in an insert update or delete SQL statement modification to data in the view will be immediately passed to the underline table.

Rules for update view :

- View define from single table.
- If the user wants to insert record with the help of view, then the primary key column and all the NOT NULL column must be include in the view.
- The user can update, delete record with the help of view even if the primary key column and NOT NULL column are excluded from the view define.

Common rules for update view (allow) :

1) The view is defined based on one and only one table.
2) The view must include the PRIMARY KEY of the table based upon which the view has been created.
3) The view should not have any field made out of aggregate functions.
4) The view must not have any DISTINCT clause in its definition.
5) The view must not have an GROUP BY or HAVING clause in its definition.
6) If the view you want to update is based upon another view, the later should be updatable.
7) Any of the selected output fields (of the view) must not use constants, strings or value expressions.

Syntax :
UPDATE <viewname>
SET <column1>=<value1>,<column2>=<value2> WHERE <condition>;

Example :
UPDATE v2
SET ename="William" WHERE eno=1;

Destroy view Syntax :
DROP VIEW <viewname>;
Example :
DROP VIEW v1;

- **Difference between View & Table**

View	**Table**
A Database object that allows generating a logical subset of data from one or more tables.	A Database object or an entity that stores the data of a database.
A virtual table.	An actual table.
View depends on table.	Table is an independent

- **Difference between Delete & Truncate**

Delete	Truncate
The DELETE command is used to delete specified rows(one or more).	While this command is used to delete all the rows from a table.
It is a DML(Data Manipulation Language) command.	While it is a DDL(Data Definition Language) command.
There may be a WHERE clause in the DELETE command in order to filter the records.	While there may not be WHERE clause in the TRUNCATE command.
In the DELETE command, a tuple is locked	While in this command, the data

Delete	Truncate
before removing it.	page is locked before removing the table data.
The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row.	TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log.
DELETE command is slower than TRUNCATE command.	While the TRUNCATE command is faster than the DELETE command.
To use Delete you need DELETE permission on the table.	To use Truncate on a table we need at least ALTER permission on the table.
The identity of the fewer column retains the identity after using DELETE Statement on the table.	Identity the column is reset to its seed value if the table contains an identity column.
The delete can be used with indexed views.	Truncate cannot be used with indexed views.
This command can also active trigger.	This command does not active trigger.
DELETE statement occupies more transaction spaces than Truncate.	Truncate statement occupies less transaction spaces than DELETE.