Made by Shravan Kumar

Submit to CSS CORP

SQL Statements

1) **Create databae statement?**

CREATE DATABASE database_name

Ex

CREATE DATABASE Shravan_db

2) **Create table ?**

```
CREATE TABLE table_name

( column_name1 data_type,

column_name2 data_type,

column_name3 data_type, .... );
```

EX

```
CREATE TABLE Room
(
Room_Id int,
LastName varchar(50),
FirstName varchar(50),
Address varchar(50,
City varchar(50)
 );
```

Constraints

1) SQL 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 (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

•NOT NULL
•UNIQUE
•PRIMARY KEY
•FOREIGN KEY
•CHECK •DEFAULT

SQL NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values.

```
CREATE TABLE Room

(

 Room_Id int NOT NULL,

 LastName varchar(255) NOT NULL,

 FirstName varchar(255),

 Address varchar(255),

  City varchar(255) );
```

SQL UNIQUE Constraint

On CREATE TABLE The following SQL creates a UNIQUE constraint on the "Room_Id" column when the "Room" table is created:

```
CREATE TABLE Room

 (

 Room_Id int NOT NULL,

 LastName varchar(255) NOT NULL,

 FirstName varchar(255),

 Address varchar(255), City varchar(255),

 CONSTRAINT uc_RoomID UNIQUE (Room_Id,LastName)

);
```

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. Each table should have a primary key, and each table can have only ONE primary key

```
CREATE TABLE Room
( Room_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_RoomID PRIMARY KEY (Room_Id,LastName) );
```

To DROP a PRIMARY KEY Constraint

use the following SQL: MySQL:

```
ALTER TABLE Persons DROP PRIMARY KEY
```

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between Tables.

EX

```
CREATE TABLE Hostel
(
H_Id int NOT NULL,
HostelNo int NOT NULL,
Room_Id int, PRIMARY KEY (H_Id),
FOREIGN KEY (Room_Id) REFERENCES Room(Room_Id)
);
```

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

To DROP a FOREIGN KEY Constraint

use the following SQL: MySQL:

```
ALTER TABLE Orders DROP FOREIGN KEY fk_PerOrders
```

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

The following SQL creates a CHECK constraint on the "Room_Id" column when the "Room" table is created. The CHECK constraint specifies that the column "Room_Id" must only include integers greater than 0.

EX

```
CREATE TABLE Room
 (
 Room _Id int NOT NULL,
LastName varchar(255) NOT NULL,
 FirstName varchar(255), Address varchar(255),
City varchar(255),
 CONSTRAINT chk_Room CHECK (Room_Id>0 AND City='Sandnes')
);
```

To DROP a CHECK Constraint

use the following SQL: MySQL:

```
ALTER TABLE Room DROP CHECK chk_Room
```

SQL DEFAULT Constraint

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

The default value will be added to all new records, if no other value is specified.

The following SQL creates a DEFAULT constraint on the "City" column when the "Room" table is created:

EX

```
CREATE TABLE Room
(
Room_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
);
```

The DEFAULT constraint can also be used to insert system values, by using functions

Like GETDATE():

```
CREATE TABLE Hostel
(
H_Id int NOT NULL,
 HostelNo int NOT NULL,
 Room_Id int,
 JionDate date DEFAULT GETDATE()
);
```

INSERT

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

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

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

EX

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

```
INSERT INTO Customers ( CustomerName, ContactName, Address, City, PostalCode,
Country)
VALUES ('shravan', 'kumar', 'hyderabad', 'hyderabad', '500005', 'india');
```

SELECT

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

The data returned is stored in a result table

EX

```
SELECT * FROM table_name;
```

```
SELECT * FROM Customers;
```

UPDATE

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

EX

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

```
UPDATE Customers
SET ContactName = 'Shravan', City= 'Hyderabad'
WHERE CustomerID = 1;
```

DELETE

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

EX

```
DELETE FROM table_name WHERE condition;
```

```
DELETE FROM Customers WHERE CustomerName='Shravan';
```

WHERE

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

EX

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

```
SELECT * FROM Customers
WHERE Country='india';
```

Wildcard

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

EX

```
SELECT * FROM Customers
WHERE City LIKE 'Hyd%';
```

Aliases.

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.

EX

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

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

JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

INNER JOIN:

EX

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

LEFT JOIN

EX

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

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

RIGHT JOIN

EX

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

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

FULL OUTER JOIN

EX

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

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