SQL DDL DML

DDL – Data Definition Language

- Modify Database Structures
- `Create`
- `Alter`
- `Drop`

DML – Data Manipulation Language

- Modify the data values within the tables/rows
- `Insert`
- `Update`
- `Delete`

SQL Insert

INSERT statement – Two Formats

```
INSERT INTO <table name> (column, column, column)
   VALUES (value, value, value)
```

(if no column & value are specified, NULL or default will be assigned)

```
INSERT INTO <table name>
   VALUES (value, value, value, value)
```

(must have a value or NULL for every column in the table)

SQL Insert

INSERT statement (with auto_increment)

```
INSERT INTO nwEmployees
    (LastName, FirstName, Title, TitleOfCourtesy,
    BirthDate, HireDate, Address, City, Region,
    PostalCode, Country, HomePhone, Extension)

    VALUES

    ('Dunn','Nat','Sales Representative', 'Mr.',
    '1970-02-19', '2014-01-15',
    '4933 Jamesville Rd.','Jamesville','NY',
    '13078','USA','315-555-5555','130');
```

INSERT statement (2nd format missing values)

```
INSERT INTO nwEmployees
   VALUES
   ('20','Thomas','Tammy','Database Administrator',
   'Ms.','1990-08-27', '2017-06-18',
   '5012 Arapahoe St.','Boulder','CO',
   '80304','USA');

INSERT INTO nwEmployees
   VALUES
   ('20','Thomas','Tammy','Database Administrator',
   'Ms.','1990-08-27', '2017-06-18',
   '5012 Arapahoe St.','Boulder','CO',
   '80304','USA',NULL,NULL,NULL,NULL,NULL);
```

CREATE statement

```
CREATE TABLE <table name>
   (column  DATATYPE(L),
    column  DATATYPE(L) NOT NULL,
    column  DATATYPE(L) NOT NULL Default 0,
    column  DATATYPE(L) CONSTRAINT
        <constraint name> TYPE,
    column  DATATYPE(L))
```

DESCRIBE statement

shows you what MySQL knows about a table

CREATE statement

```
CREATE TABLE IF NOT EXISTS items (
    itemID      INT           NOT NULL AUTO_INCREMENT,
    itemCode    CHAR(3)       ,
    itemname    VARCHAR(40) NOT NULL DEFAULT ' ',
    quantity    INT           NOT NULL DEFAULT 0,
    price       DECIMAL(9,2)NOT NULL DEFAULT 0,
    PRIMARY KEY  (itemID)
         );

DROP TABLE IF EXISTS items;

DESC items;
```

CREATE statement

```
CREATE TABLE items (
    itemID      INT           NOT NULL ,
    itemCode    CHAR(3)       ,
    itemname    VARCHAR(40)   NOT NULL DEFAULT ' ',
    quantity    INT           NOT NULL DEFAULT 0,
    price       DECIMAL(9,2) NOT NULL DEFAULT 0,
    PRIMARY KEY  (itemID)
         );
```

TRUNCATE statement – removes all rows, keeps structure

```
TRUNCATE TABLE <table name>
```

DROP statement -- removes all rows, removes structure

```
DROP TABLE <table name>
```

ALTER statement

```
ALTER TABLE <table name>
   ADD/MODIFY/DROP
          COLUMN <column name>  DATATYPE(L),

   RENAME <new table name>

ALTER TABLE <table name>
   DROP COLUMN
```

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-92233720368547758 08 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

http://www.mysqltutorial.org/mysql-data-types.aspx

ENUM – entering a small array into a single column

http://www.mysqltutorial.org/mysql-enum/

```
2  CREATE TABLE tickets (
3      id INT PRIMARY KEY AUTO_INCREMENT,
4      title VARCHAR(255) NOT NULL,
5      priority ENUM('Low', 'Medium', 'High') NOT NULL
   );
```

```
1  INSERT INTO tickets(title, priority)
2  VALUES('Scan virus for computer A', 'High');
```

```
1  INSERT INTO tickets(title, priority)
2  VALUES('Upgrade Windows OS for all computers', 1);
```

MySQL spatial data types

MySQL supports many spatial data types that contain various kinds of geometrical and geographical values as shown in the following table:

Spatial Data Types	Description
`GEOMETRY`	A spatial value of any type
`POINT`	A point (a pair of X-Y coordinates)
`LINESTRING`	A curve (one or more `POINT` values)
`POLYGON`	A polygon
`GEOMETRYCOLLECTION`	A collection of `GEOMETRY` values
`MULTILINESTRING`	A collection of `LINESTRING` values
`MULTIPOINT`	A collection of `POINT` values
`MULTIPOLYGON`	A collection of `POLYGON` values

SQL Insert

JSON

```
1 CREATE TABLE events(
2   id int auto_increment primary key,
3   event_name varchar(255),
4   visitor varchar(255),
5   properties json,
6   browser json
7 );
```

The `properties` and `browser` columns are the JSON columns. They are used to store properties of an event and specification of the browser that visitors use to browse the website.

SQL Insert

Let's insert some data into the `events` table:

```
INSERT INTO events(event_name, visitor,properties, browser)
VALUES (
  'pageview',
   '1',
   '{ "page": "/" }',
   '{ "name": "Safari", "os": "Mac", "resolution": { "x": 1920, "y": 1080 } }'
),
('pageview',
  '2',
  '{ "page": "/contact" }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 2560, "y": 1600 }
}'
),
(
  'pageview',
  '1',
  '{ "page": "/products" }',
  '{ "name": "Safari", "os": "Mac", "resolution": { "x": 1920, "y": 1080 } }'
),
(
  'purchase',
   '3',
  '{ "amount": 200 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1600, "y": 900 }
}'
),
(
  'purchase',
   '4',
  '{ "amount": 150 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1280, "y": 800 }
}'
),
```

ALTER statement

```
ALTER TABLE nwemployees
   MODIFY COLUMN EmployeeID INT(11) PRIMARY KEY
   AUTO_INCREMENT;

ALTER TABLE Items
   ADD PRIMARY KEY (ItemID) ;

ALTER TABLE Items
   ADD COLUMN InventoryDate DATE AFTER itemname ;

ALTER TABLE Items
   DROP COLUMN InventoryDate;
```

BULK INSERT statement

```
INSERT INTO items
   SELECT ProductID, CategoryID,  ProductName,
   CURDATE(), unitsInStock, UnitPrice
      FROM nwProducts
   ;
```

UPDATE statement

```
UPDATE <table name>
   SET column = <value>
   WHERE <condition>
```

DELETE statement

```
DELETE FROM <table name>
   WHERE <condition>
```

Note: Without the WHERE clause,
the DELETE will affect ALL rows

UPDATE statement

```
UPDATE items
   SET price = (price + (price * .05))
   WHERE itemcode = 1;

UPDATE items
   SET price = ROUND((price + (price * .05)),2)
   WHERE itemcode = 1;
```

Delete statement

```
DELETE FROM items
   WHERE itemcode = 2;
```

The VIEW

- A “VIEW” is an empty shell of a table definition
- The view contains no data until it is queried
- Sometimes considered a “Virtual Table”
- Each time the view is queried, the underlying query that populates the view is re-executed

CREATING a VIEW

```
CREATE VIEW <view name> AS
   SELECT <col1>, <col2>, <col3>
      FROM <table1>
      WHERE <condition>
```

Why VIEWs?

- The base table or specific columns in the base table can be hidden from certain users who are only allowed access to the view
- Very complex SQL to create the view can be hidden from end users

First "why":

Base Table:

Employees(EmpID, Lastname, Firstname, Salary, HireDate)

View:

Employees(EmpID, Lastname, Firstname, HireDate)

Second "why":

Base Query:

```
Create VIEW TopEmployeeOrders AS
   Select LastName, Firstname,
    sum(UnitPrice * Quantity) as 'OrderValue'
   from nwEmployees E, nwOrders O,
     nwOrderDetails D
   where E.EmployeeID = O.EmployeeID
     and O.OrderID = D.OrderID
   GROUP BY LastName, FirstName
   Order By 3 desc
```

View

```
Select * from TopEmployeeOrders;
```

```
CREATE OR REPLACE VIEW TopEmployeeOrders AS
   SELECT LastName, Firstname,
    SUM(UnitPrice * Quantity) AS 'OrderValue'
   FROM nwEmployees E, nwOrders O,
     nwOrderDetails D
   WHERE E.EmployeeID = O.EmployeeID
     AND O.OrderID = D.OrderID
   GROUP BY LastName, FirstName
   ORDER BY 3 DESC;

SELECT * FROM TopEmployeeOrders;
```