SQL: Data Manipulation and View

Prof. Hyuk-Yoon Kwon
https://sites.google.com/view/seoultech-bigdata

Most parts are based on slides used in Stanford (http://web.stanford.edu/class/cs145)

Today’s Lecture

- **Today’s lecture**
  - Data manipulation
  - View

INSERT

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

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

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

Demo Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

INSERT Example

- The following SQL statement inserts a new record in the "Customers" table:

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

- Insert Data Only in Specified Columns

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

Insert Into Select

The INSERT INTO SELECT Statement

- The INSERT INTO SELECT statement copies data from one table and inserts it into another table.
  - INSERT INTO SELECT requires that data types in source and target tables match
  - The existing records in the target table are unaffected

INSERT INTO SELECT Syntax

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

Demo Database

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

SupplierID	SupplierName	ContactName	Address	City	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

Insert Into Select Example

- The following SQL statement copies "Suppliers" into "Customers"

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

- The following SQL statement copies "Suppliers" into "Customers"

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;
```

- The following SQL statement copies only the German suppliers into "Customers"

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

Insert ALL

INSERT ALL statement

- To add multiple rows with a single INSERT statement

```
INSERT ALL
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
SELECT * FROM dual;
```

Example – Insert into single table

```
INSERT ALL
  INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM')
  INTO suppliers (supplier_id, supplier_name) VALUES (2000, 'Microsoft')
  INTO suppliers (supplier_id, supplier_name) VALUES (3000, 'Google')
SELECT * FROM dual;
```

This is equivalent to the following 3 INSERT statements:

```
INSERT INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM');

INSERT INTO suppliers (supplier_id, supplier_name) VALUES (2000, 'Microsoft');

INSERT INTO suppliers (supplier_id, supplier_name) VALUES (3000, 'Google');
```

Example – Insert into multiple tables

```
INSERT ALL
  INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM')
  INTO suppliers (supplier_id, supplier_name) VALUES (2000, 'Microsoft')
  INTO customers (customer_id, customer_name, city) VALUES (999999, 'Anderson Construction', 'New York')
SELECT * FROM dual;
```

This example will insert 2 rows into the *suppliers* table and 1 row into the *customers* table. It is equivalent to running these 3 INSERT statements:

```
INSERT INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'IBM');

INSERT INTO suppliers (supplier_id, supplier_name) VALUES (2000, 'Microsoft');

INSERT INTO customers (customer_id, customer_name, city) VALUES (999999, 'Anderson Construction', 'New York');
```

Practice#1: INSERT Data with a Condition

- When we insert data from a table Suppliers into another table Customers, how do we make sure that we do not enter the same customer information again? Here, the same customer information means that customer ID is the same as supplier ID.
  - Hint: Use Nested query with NOT EXISTS()

Update

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

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

Update Example

- The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

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

- UPDATE Multiple Records

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

Update table with data from another table

```
UPDATE customers
SET c_details = (SELECT contract_date
                 FROM suppliers
                 WHERE suppliers.supplier_name = customers.customer_name)
WHERE customer_id < 1000;
```

Using EXISTS Clause

```
UPDATE suppliers
SET supplier_name = (SELECT customers.customer_name
                     FROM customers
                     WHERE customers.customer_id = suppliers.supplier_id)
WHERE EXISTS (SELECT customers.customer_name
              FROM customers
              WHERE customers.customer_id = suppliers.supplier_id);
```

Delete

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

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

Delete Example

- The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

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

- Delete All records

```
DELETE FROM Customers;
```

- Using EXISTS clause

```
DELETE FROM suppliers
WHERE EXISTS
  ( SELECT customers.customer_name
    FROM customers
    WHERE customers.customer_id = suppliers.supplier_id
    AND customer_id > 25 );
```

CREATE TABLE

- CREATE TABLE Statement
  - The CREATE TABLE statement is used to create a new table in a database

- Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Create Table Example

- The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City

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

- NOT NULL

```
CREATE TABLE customers
(
  customer_id integer NOT NULL,
  customer_name varchar(50) NOT NULL,
  city varchar(50)
);
```

Create Table Using Another Table

- A copy of an existing table can also be created using CREATE TABLE

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

- The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

Practice #2: Create Table

- How can we create a table from another table without copying any values from the old table?

Drop Table

- DROP TABLE Statement
  - The DROP TABLE statement is used to drop an existing table in a database.

- **Syntax**

```
DROP TABLE table_name;
```

Drop Table Example

- The following SQL statement drops the existing table "Shippers":

```
DROP TABLE Shippers;
```

- TRUNCATE TABLE

```
TRUNCATE TABLE table_name;
```

Alter Table

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

```
ALTER TABLE table_name
ADD column_name datatype;
```

  - The following SQL adds an "Email" column to the "Customers" table:

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

ALTER TABLE - DROP COLUMN

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

- The following SQL deletes the "Email" column from the "Customers" table:

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

ALTER TABLE - MODIFY COLUMN

```
ALTER TABLE table_name
MODIFY column_name datatype;
```

Example – Modify datatype of each column

ID	LastName
1	
2	
3	

ALTER TABLE - RENAME

```
ALTER TABLE table_name
  RENAME COLUMN old_name TO new_name;
```

```
ALTER TABLE table_name
  RENAME TO new_table_name;
```

Select Top

SELECT TOP Clause

- The SELECT TOP clause is used to specify the number of records to return.

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

- The following SQL statement selects the first three records from the "Customers" table:

```
SELECT * FROM Customers
WHERE ROWNUM <= 3;
```

Views

- In some cases, it is not desirable for all users to see the entire logical model
- Consider a person who needs to know an employee name and department, but not the salary. This person should see a relation described, in SQL, by

  select *name*, *dept_name*

  from *Employee*

- A view provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a view.

View Definition

- **A view is defined using the create view statement which has the form**

  create view *v* as < query expression >

  where <query expression> is any legal SQL expression. The view name is represented by *v*.

- **Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.**

- **View definition is not the same as creating a new relation by evaluating the query expression**
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

Example Views

- **A view of Employee without their salary**

  create view *Employee_no_salary* as

  select *name*, *dept_name*

  from *Employee*

Name	Dept_name	salary
Alice	Biology	2000
Bob	Business	3000
Chen	Biology	3500

- **Find all instructors in the Biology department**

  select *name*

  from *Employee_no_salary*

  where *dept_name* = 'Biology'

- **Create a view of department salary totals**

  create view *departments_total_salary*(*dept_name*, *total_salary*) as

  select *dept_name*, sum (*salary*)

  from *Employee*

  group by *dept_name*;

Views Defined Using Other Views

- create view *physics_fall_2009* as

  select *course.course_id*, *sec_id*, *building*, *room_number*

  from *course*, *section*

  where *course.course_id* = *section.course_id*

  and *course.dept_name* = 'Physics'

  and *section.semester* = 'Fall'

  and *section.year* = '2009';

- create view *physics_fall_2009_watson* as

  select *course_id*, *room_number*

  from *physics_fall_2009*

  where *building*= 'Watson';

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to *depend on* view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be *recursive* if it depends on itself.