SQL: Data Manipulation and View

Prof. Hyuk-Yoon Kwon

https://sites.google.com/view/seoultech-bigdata

Today's lecture

- Data manipulation
- View

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

Select Top (DML)

: 특정 record 수 / 범위에 따라 결과 filtering 시 사용
(Oracle, Mysql 에서는 LIMIT)

SELECT TOP Clause

- The SELECT TOP clause is used to specify the number of records to return.
- Selects the first three records from the "Customers" table:

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

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

INSERT (DML)

The INSERT INTO Statement

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

INSERT INTO : 새로운 Record 추가에 사용
Table 이름 지정 후 새로운 record에 들어갈 data 제공

INSERT INTO Syntax

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

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

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 (DML)

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
: Table에서 data 선택 & 다른 table에 삽입 시 사용
-> data 선택 위한 SELECT 문 작성 후 결과를
   INSERT INTO 절 다음 목적지 TABLE 이름과 함께 지정
⇒ SELECT문에서 선택된 data가 새로운 table에 추가됨

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;
```

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

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

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

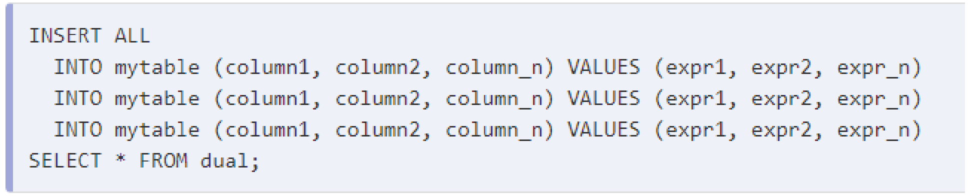
The following SQL statement copies only the German suppliers into "Customers"

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

Insert ALL (DML)

INSERT ALL statement

- To add multiple rows with a single INSERT statement



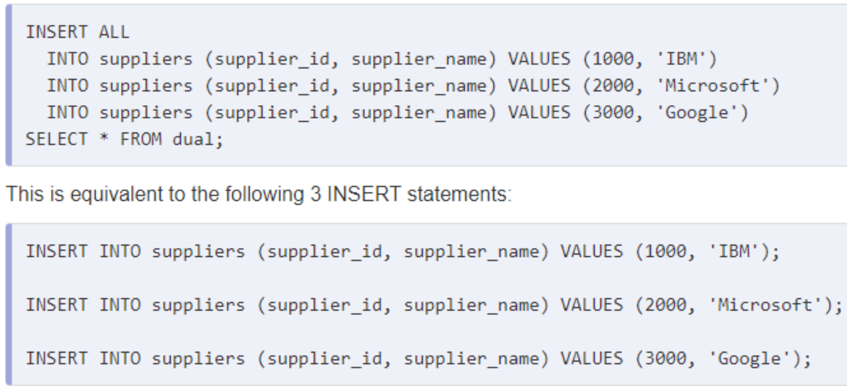
```
INSERT ALL
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO mytable (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
SELECT * FROM dual;
```

INSERT ALL

: 여러 table 에 data 삽입 시 사용

-> 하나의 INSERT 문으로 여러 table에 data 삽입

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;
```

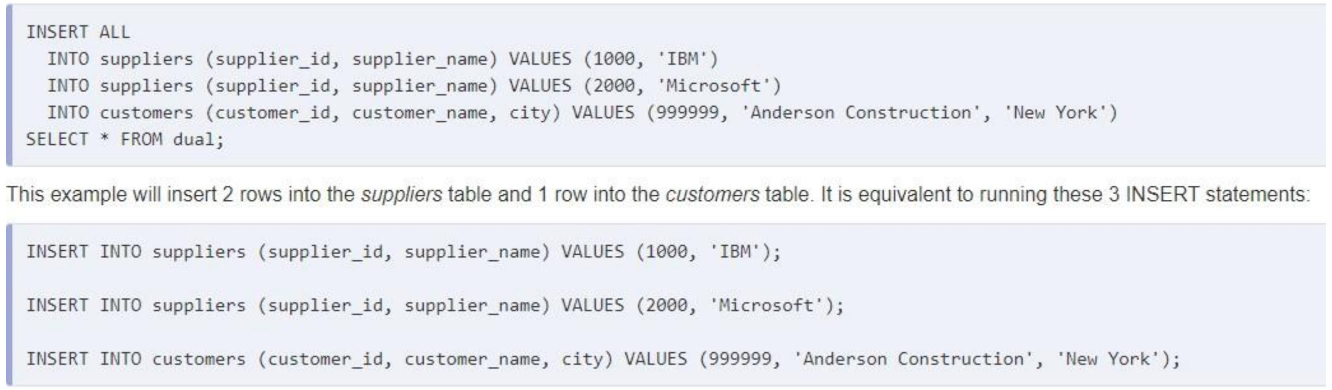
This is equivalent to the following 3 INSERT statements:

```
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');
```

Update (DML)

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 Multiple Records

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

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 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 (DML)

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

- DELETE Syntax

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

WHERE 절은 선택적으로 사용 가능
(X 사용 시 해당 table의 모든 행 삭제)

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

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

- Delete All records

```sql
DELETE FROM Customers;
```

- Using EXISTS clause

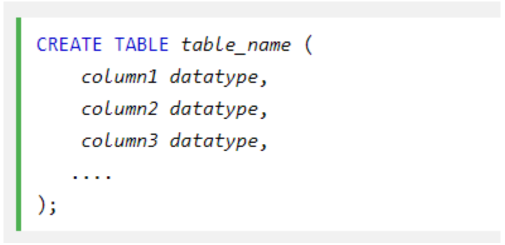
```sql
DELETE FROM suppliers
WHERE EXISTS
  ( SELECT customers.customer_name
    FROM customers
    WHERE customers.customer_id = suppliers.supplier_id
    AND customer_id > 25 );
```

CREATE TABLE (DDL)

CREATE : DB Object 생성 시 사용
ex) DB, Table, View, Index, Procedure ...
-> 해당 객체 유형 따라 CREATE 구문 다를 수 있음

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

- Syntax



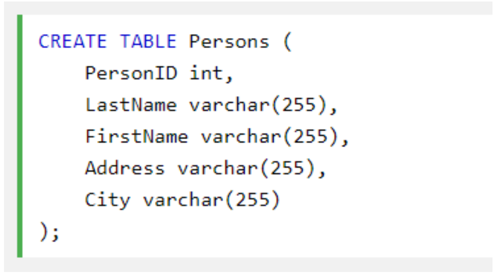
```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

- NOT NULL

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

- The following example creates a table called "Persons" that contains five columns

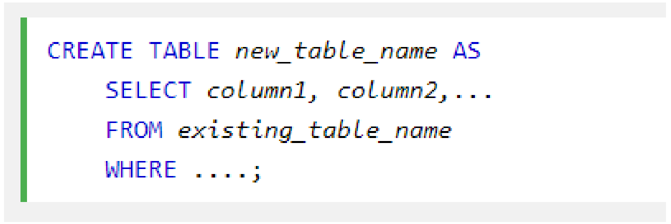
  : PersonID, LastName, FirstName, Address, and City



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

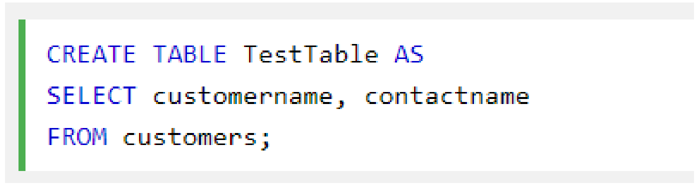
Create Table Using Another Table

- 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;
```

Drop Table (DDL)

DROP : Table, View, Index ... 의 객체 삭제 시 사용

DROP TABLE Statement

=> Table과 그에 연관된 모든 Index, Constraints, Triggers, ... 의 객체가 DB에서 완전히 제거됨

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

Syntax

```
DROP TABLE table_name;
```

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

```
DROP TABLE Shippers;
```

TRUNCATE TABLE

```
TRUNCATE TABLE table_name;
```

TRUNCATE TABLE

특정 Table의 모든 행 삭제 시 사용

-> 모든 data 삭제 = 초기화 (table은 그대로 유지)

=> 삭제되는 data가 유용 / 재사용 가능성 있는 경우 DELETE 문이 안전

-> Table의 Schema, Index, Constraints, ... 이 유지됨

-> DELETE 보다 더 빠르게 작동

Alter Table (DDL)

ALTER TABLE
: 이미 존재하는 Table 구조 변경 시 사용
-> 열 추가 / 삭제, 열 data type 변경, constraints 추가 / 삭제 , ...
-> DB Schema 변경 시 유용

ALTER TABLE Statement

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

ALTER TABLE - ADD Column

- adds an "Email" column to the "Customers" table:

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

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

ALTER TABLE - DROP COLUMN

- deletes the "Email" column from the "Customers" table:

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

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

ALTER TABLE - MODIFY COLUMN

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

MODIFY (ALTER TABLE 문과 함께 사용)
: Table 열 수정 시 Data type / 크기 / Attribute 변경 시 사용

ALTER TABLE - RENAME

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

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

Views

Logic Model : DB 내 여러 Table (관련 있을 수도 없을 수도)

View : 특정 열 / 행 선택하여 가상의 Table 만들어 사용 가능
-> Data 가시성 제어 기능 (data 일부분만 보기 , 불필요한 data 숨기기)
-> 여러 Table을 Join하여 1개의 가상 table로 제작 가능
-> 실제 data 저장 X
⇒ DB 보안성 증가 & 복잡성 감소 & data에 대한 간단한 interface 제공

- 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

View 정의 : 표현식 저장 & 해당 표현식이 view를 사용하는 query에 삽입됨
(새로운 관계 평가-생성 X)
-> View가 정의되면 해당 이름으로 생성된 가상 관계 참조 가능

- 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.

Views Defined Using Other Views

- One view may be used in the expression defining another view

View는 다른 View의 표현식에서 사용될 수 있음
(View가 기존 table / 다른 view를 기반으로 생성되는 가상의 table이므로 가능)

- A view relation *v* is said to be *recursive* if it depends on itself.

View는 자기 자신에게 의존 가능
Ex) 하나의 view가 자신을 참조 / 자기 자신에 대한 조건 포함 가능

*** View Example**

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

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

```
create view Employee_no_salary as
select name, dept_name
 from Employee
```

- **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;
```

- 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

Dependency = View 간의 의존성
1) v1이 v2에 직접적으로 의존 : "v1은 v2에 직접 의존한다"
2) v1에서 v2에 이르는 의존성 경로 존재 : "v1 은 v2에 의존한다"
=> DB system에서 View 관리 & 최적화 시 중요한 역할

```
■ 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';
```