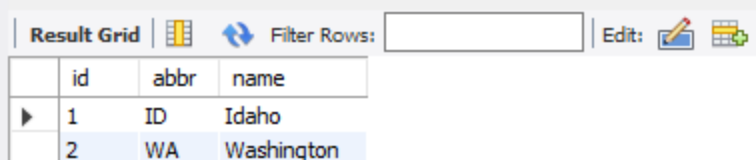
Dynamic Foreign Key Inserts

- It is bad practice to insert static foreign keys into your database because it is often difficult to know what the corresponding primary key will be.
- Dynamic foreign key inserts are especially useful when writing scripts to populate database tables. Without them, it would be almost impossible to populate a large database.
- Without dynamic foreign key inserts, you would have to look up the corresponding primary key before you could insert data into a table that has a foreign key column.

Static Foreign Key Example

In the University Database assignment week 11 there is a states table, and a student table with a foreign key column to the states table. Before we can insert a value into the student table, there must first be values in the states table. Below is an example of two states that have been inserted.



id	abbr	name
1	ID	Idaho
2	WA	Washington

Now, we can create an INSERT statement to populate the student table. Below is an example of a static foreign key INSERT which it the WRONG way to do it.

```
INSERT INTO `students`
      (`fname`, `lname`, `gender`, `city`, `state_id`, `dob`)
VALUES ('Jim', 'Smith', 'm', 'Rexburg', '1', '2000-01-01');
```

- Notice that the static key of '1' was inserted as the foreign key for Idaho from the states table. This is bad practice because when you write a script, you have no way of knowing what that foreign key will be.
- The correct way to code the INSERT statement is to set the states auto-increment primary key value to a variable after each row is inserted. Then when inserting values into the students table you can use the variables created during the insertion of rows into the states table.

Below is an example:

- INSERT a row into the states table.

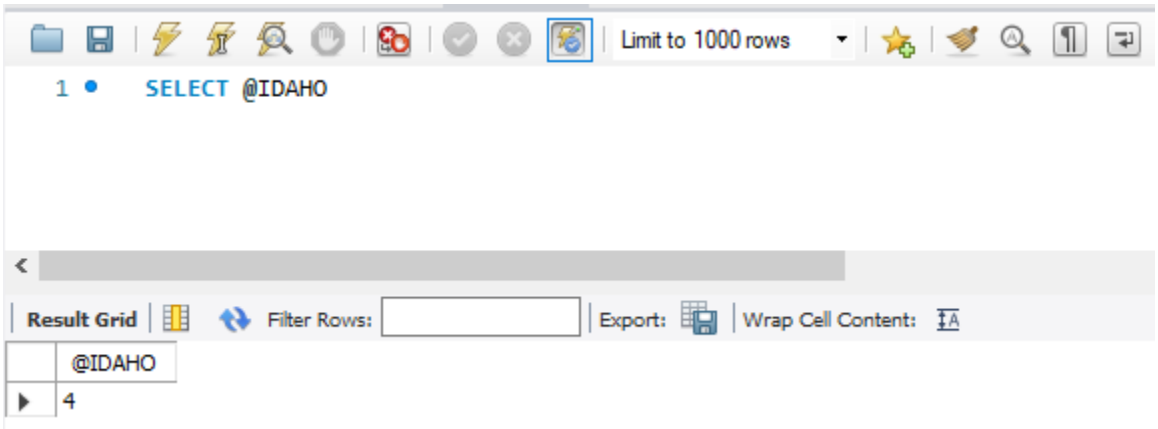
```
INSERT INTO `enrollment`.`states` (`abbr`, `name`) VALUES ('ID',
'Idaho');
```

- Notice that we did not explicitly assign a value to the id field. This is because the field is auto-increment, so the system will provide a value for us.

- We don't know what the value is of the assigned id, but we can get the value be running the **last_insert_id()** function and assigning that value to a variable.

```
SET @Idaho = last_insert_id();
```

- This variable is only valid during the current SQL session. If we logout, we no longer have access to that variable.
- If we want to see the value that was assigned to the variable we can run a SELECT statement



- Notice that the auto-increment value assigned to the @Idaho variable was 4 and not 1. This is because I had previously created rows in the states table, and then deleted them. The auto-increment column never re-uses values that have been used previously. Instead, the RDMBS keeps track of the last auto-increment key and assigns the next highest value when a new row is inserted.
- We would repeat this process in our script for each row of the states table. We insert a row, then assign the **last_insert_id()** to a variable.

Example:

```
INSERT INTO `enrollment`.`states` (`abbr`, `name`) VALUES ('WA',
'Washington');
SET @Washington = last_insert_id();

INSERT INTO `enrollment`.`states` (`abbr`, `name`) VALUES ('UT',
'Utah');
SET @Utah = last_insert_id();
```

- Once you have inserted data into all tables with no foreign key dependencies and set variables for primary keys, you can begin inserting data into tables with foreign keys using the variables you created previously for their corresponding primary key.

Example:

```
INSERT INTO `students`
      (`fname`, `lname`, `gender`, `city`, `state_id`, `dob`)
VALUES ('Jim', 'Smith', 'm', 'Rexburg', '@Idaho', '2000-01-01');

INSERT INTO `students`
      (`fname`, `lname`, `gender`, `city`, `state_id`, `dob`)
VALUES ('Jane', 'Martinez', 'f', `Provo`, '@Utah', '2001-02-05');

INSERT INTO `students`
      (`fname`, `lname`, `gender`, `city`, `state_id`, `dob`)
VALUES ('George', 'Benson', 'm', `Seattle`, '@Washington', '1960-
04-05');
```