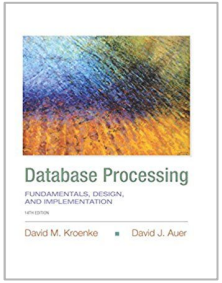
Fairfield University
DOLAN
School of Business



Databases for Analytics

Kroenke / Auer
Chapter 7 (partial)

Learning Objectives

- **Skills:** You should know how to ...
  - Write SQL DML code to INSERT, UPDATE, DELETE table rows
  - Use subselect queries to populate / update table data
- **Theory:** You should be able to explain ...
  - The effect of referential integrity on table load/delete order
  - The logic of SQL transactions and procedures

Basic SQL DML

INSERT, UPDATE, DELETE

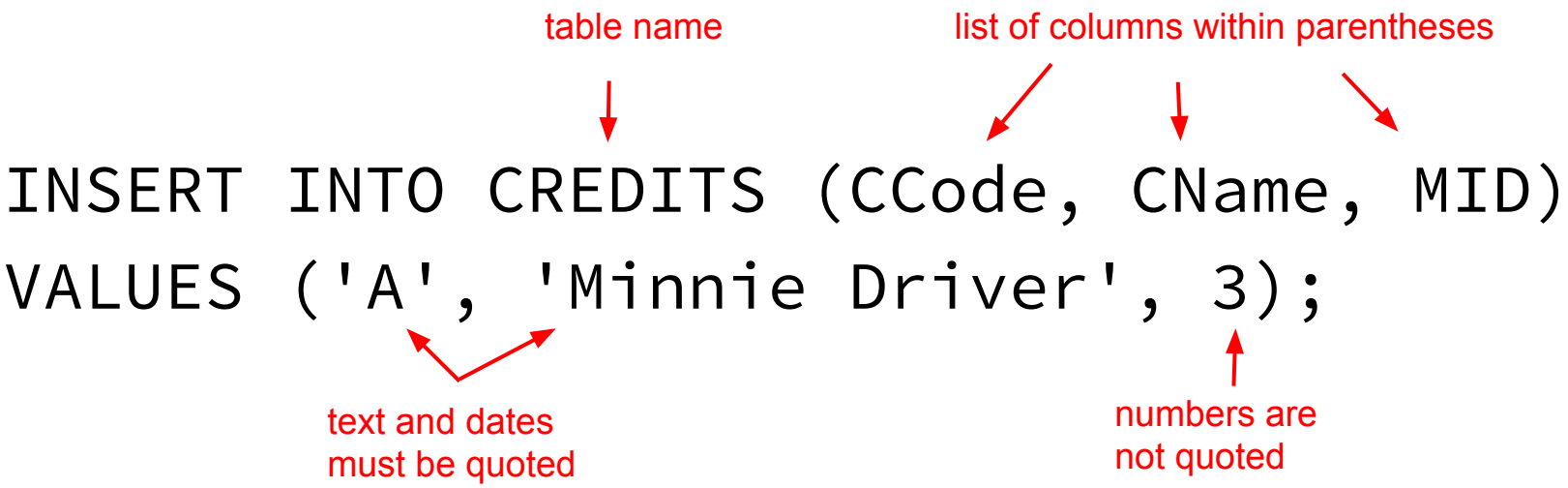
The `INSERT INTO` Command

ANSI Standard Syntax:

`INSERT INTO` *table* **(***column1, column2, …, columnN***)**

`VALUES (`*value1*, *value2*, … , *valueN*`);`**

INSERT INTO Example



The number of values must match
number of columns; can use NULL
unless a constraint forbids it

Shortcut for Multiple Records

```
INSERT INTO CREDITS (CCode, CName, MID)
VALUES ('A', 'Minnie Driver', 3),
       ('A', 'Tony Shaloub', 3),
        … ;
```

The same columns are used for each VALUES tuple.

INSERT INTO with a Subquery

- Inserting rows from a SELECT:

  INSERT INTO *table* **(***columns***)**

  SELECT *columns* **FROM** *tables* **…;**

- Example:

```
INSERT INTO MOVIES (Title,Rating)
   SELECT DISTINCT MTitle, Rating
   FROM DATASET;
```

CREATE TABLE with a Subquery

CREATE TABLE *table*

SELECT *columns* **FROM** *tables* …**;**

- This is a DDL/DML mashup. Column definitions are derived from the subquery.
- Useful to create temporary tables from existing data, usually as a backup or to speed up a multi-step transaction.

The `UPDATE` Command

```
UPDATE table
SET
    column1 = value1, column2 = value2, ...
WHERE
    conditions;
```

- Used to modify existing records

UPDATE with JOINs

```
UPDATE table1 JOIN table2 ON (...)
SET
    column values from any of the joined tables
WHERE
    conditions;
```

- Can update columns from multiple tables
- Conditions can mix columns from multiple tables

UPDATE with a Subquery

```
UPDATE table
SET
    column1 = (subquery), ...
WHERE
    conditions;
```

- The subquery must return a single scalar value
- We can also use subqueries in the `WHERE` clause

The `DELETE` Command

```
DELETE FROM table
WHERE conditions;
```

- Deletes selected records/rows from a single table
- The `WHERE` clause can include a subquery if needed

DELETE with JOINs

DELETE *table1* ← Table with records to delete

FROM *table1* **JOIN** *table2* **ON (...)**

WHERE *conditions*; ← Conditions can use data from any table in the FROM clause

This one is a bit tricky and vendor-specific. Use at your own risk.

- Can delete from multiple tables if needed. Just add the second table to the `DELETE` clause.
- To delete from *all* referenced tables just omit the table names from the `DELETE` clause.

The TRUNCATE Command

TRUNCATE TABLE *table*;

- Deletes all records from a table, while also resetting any surrogate key (auto-increment) counters.
- Technically this is a DDL command, but is included here because it deletes data.

Referential Integrity Effects

The implications of mandatory relationships

INSERT Order

When writing new records to tables, always try to load the parent (one) side of each relationship first.

So general practice is to …

1. Load data into the strong entities.
2. Load data into 1-st degree child entities.
3. Load data into 2-nd degree grandchild entities.

...

`DELETE` Order

When deleting records, the order is reversed from inserting them.

Delete the children (many) before deleting the parents (one).

You can avoid this problem by setting the **`ON DELETE CASCADE`** option of the FK constraint.

Disabling FK Constraints

When bulk loading a lot of data, it is faster to disable referential integrity constraints *temporarily*. Only do this if you are **100%** sure the constraints will work after you are done loading the data.

For example, in MySQL:

```
SET @@foreign_key_checks = 0;
```

... Unsafe SQL statements that work eventually…

```
SET @@foreign_key_checks = 1;
```

Transactions

Multi-step revocable actions that can affect data

Transactions

- A transaction is a set of commands that are treated as a unit
- All changes to DB are temporary until the transaction is complete
  - Can defer constraint checks (e.g., FK referential integrity) until the end of transaction
- Three commands:
  - **START TRANSACTION**, **COMMIT, ROLLBACK**

Why We Need Transactions

What happens if ...

Table A has a mandatory relationship to Table B

Table B has mandatory relationship to Table A

- Can't add the first row to A without adding a row to B *at the same time*, which is impossible if we do the `INSERT` statements one at a time.
- Solution: use a **transaction**

`START TRANSACTION` Command

`START TRANSACTION;`

- Initiates a transaction
- All changes are temporary until transaction is complete; this allows us to defer integrity checks until we are done.
- In MySQL, **`START TRANSACTION`** disables `AUTOCOMMIT` for the duration of the transaction

`COMMIT` Command

COMMIT;

- Triggers an error check to see if any constraints are violated
- Makes any changes since `START TRANSACTION` permanent (if there are no errors)
- DBMS can be configured to `AUTOCOMMIT` after each command is issued

`ROLLBACK` Command

```
ROLLBACK;
```

- Discards changes since last `COMMIT`
- **Not usually called manually**
  - If a `COMMIT` violates a constraint then the DBMS will call `ROLLBACK` to prevent corrupting the database
  - When issuing SQL commands interactively, ROLLBACK may be used as a kind of undo

The Transaction Pattern

```
START TRANSACTION;
```

… some SQL commands ...

```
COMMIT;
```

- The SQL commands are executed in the order given but not committed until the end
- **`ROLLBACK`** acts as a contingency if the SQL commands are unsuccessful

Example: Twinned INSERTs

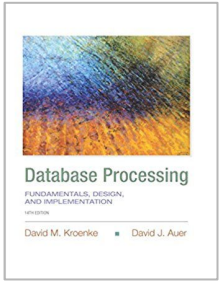
```
-- Tables A & B are in a 1:1 mandatory relationship
START TRANSACTION;
    -- AUTOCOMMIT is disabled during transactions
    INSERT INTO A (Col1,Col2) VALUES ('X','Y');
    -- A.ID is the PK for table A
    -- LAST_INSERT_ID() is last auto-increment value
    INSERT INTO B (ID,...) VALUES (LAST_INSERT_ID(),…);
COMMIT;
```

Note: LAST_INSERT_ID() is MySQL-specific.
SQLite uses LAST_INSERT_ROWID() instead.

Stored Procedures

- **Procedures** make transactions repeatable and parametric:
  - Bundle together a series of statements to be called again and again.
  - Variables are used to allow data to carry forward from one statement to another.
  - If an error occurs then the procedure can be rolled back just like a normal transaction.
- **Beyond our scope:** usually requires user permissions that are not given to analysts

Fairfield University
DOLAN
School of Business



Databases for Analytics

Kroenke / Auer
Chapters 7 (partial)