# Tools & Models for Data Science

## SQL DDL and DML

Risa Myers

Rice University

# DML & DDL

- Data Manipulation Language
  - Data retrieval
- Data Definition Language
  - Used to specify the database schema
  - Generates a schema / data dictionary
- Data Manipulation Language
  - Data insertion
  - Data deletion
  - Data modification

# DDL

- CREATE TABLE
- DROP TABLE
- TRUNCATE TABLE
- ALTER TABLE

# CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] } <tableName>
   (
      <table element list>
    );
```

# CREATE TABLE Example

```
CREATE TABLE Student
(
        netId VARCHAR(15) NOT NULL,
        lastName VARCHAR(100) NOT NULL,
        firstName VARCHAR(100) NOT NULL,
        dateOfBirth DATE NULL,
        PRIMARY KEY (netId)
);
```

# Some Words on Notation

```
CREATE TABLE Student
(
        netId VARCHAR(15) NOT NULL,
        lastName VARCHAR(100) NOT NULL,
        firstName VARCHAR(100) NOT NULL,
        dateOfBirth DATE NULL,
        PRIMARY KEY (netId)
);
```

1. Tables are often named by singular nouns
2. Table and attribute names are usually case-insensitive
3. Key constraints are typically specified after all attributes
4. Semicolons are used to separate SQL statements

- Create a table from an existing table

```
CREATE TABLE <newTableName> [AS]
        (<SELECT statement>);
```

# CREATE ... SELECT Notation

```
CREATE TABLE <newTableName> [AS]
        (<SELECT statement>);
```

- <newTableName> must **NOT** already exist in the database
- [AS] is optional
- SELECT statement may be as complex as you like

The database creates the same attribute name and types as the source table

# CREATE ... SELECT Example

Course

| crn | courseName | schedule | startTime | endTime |
|---|---|---|---|---|
| 23950 | COMP 533 | TR | 04:00 PM | 05:15 PM |
| 10626 | COMP 140 | TR | 10:50 AM | 12:05 PM |
| 16670 | COMP 430 | MWF | 02:00 PM | 02:50 PM |

```
CREATE TABLE TRCourse AS
        (SELECT *
         FROM Course
         WHERE schedule = 'TR');
```

TRCourse

| crn | courseName | schedule | startTime | endTime |
|---|---|---|---|---|
| 23950 | COMP 533 | TR | 04:00 PM | 05:15 PM |
| 10626 | COMP 140 | TR | 10:50 AM | 12:05 PM |

# CREATE ... SELECT Example 2

COURSE(CRN, COURSENAME, SCHEDULE, STARTTIME, ENDTIME)
STUDENT(NETID, FIRSTNAME, LASTNAME)
ENROLL(NETID, CRN)
SESSION(SESSIONDATE)

```
CREATE TABLE Attendance AS
SELECT lastName, firstName, sessionDate, '' AS present
FROM Course c
        JOIN Enroll e ON c.crn = e.crn
        JOIN Student s ON e.netId = s.netId
        CROSS JOIN session ss;
```

# `CREATE ... SELECT` Uses

1. Create a template and then update values
2. Perform some expensive calculation (e.g. Jaccard Index) and save the value for reuse

```
DROP TABLE [IF EXISTS] <tableName>;
```

1. Removes contents of <tableName> AND its definition
2. Why bother with `IF EXISTS`?

# Dropping Dependent Tables

? What happens if you try to drop a table that has foreign key references? (that is, other tables reference it)

# TRUNCATE

```
TRUNCATE TABLE <tableName>;
```

- Very fast
- Use with caution
  - No `DELETE` trigger is activated
  - Space is reclaimed in postgres

# Questions

1. You can create a table based on the structure and content of another table
2. `CREATE ... SELECT` overwrites the existing table
3. `TRUNCATE TABLE` should be used with caution

# Digression - Replaying Commands

- Can be a quick way to "restore" a database to a pristine condition
- Creates a repeatable process
- Ensures you have fully documented what you did
- Keeps track of the data manipulation steps
- Keeps track of schema manipulation steps

```
1  DROP TABLE IF EXISTS rates;
2  DROP TABLE IF EXISTS likes;
3  DROP TABLE IF EXISTS frequents;
4  DROP TABLE IF EXISTS serves;
5
6  CREATE TABLE IF NOT EXISTS   likes (
7  drinker VARCHAR(50),
8  coffee VARCHAR(50),
9  PRIMARY KEY(drinker, coffee)
10 );
11
12 CREATE TABLE IF NOT EXISTS   frequents
13 drinker VARCHAR(50),
14 bar VARCHAR(50),
15 PRIMARY KEY(drinker, bar)
16 );
17
18 CREATE TABLE IF NOT EXISTS   serves (
19 bar VARCHAR(50),
20 coffee VARCHAR(50),
21 PRIMARY KEY(bar, coffee)
22 );
23
24 CREATE TABLE IF NOT EXISTS   rates (
25 drinker VARCHAR(50),
26 coffee VARCHAR(50),
27 score integer
28 );
```

# ALTER TABLE

1. Add / delete columns
2. Rename table / columns
3. Add / delete constraints
4. Change column data type
   - Some systems import everything as characters
   - This can be a best practice: import as text, convert to “correct” data type
5. Change keys
6. Typically, CANNOT change attribute order

? Why import data as text?

? When might you change a key?

When might you alter a table?

- When there is additional data to store
- When you need to differentiate data
- When you learn something new about the data
- When you have to correct for defaults (Load data in SQL Server)
- When you want to disambiguate a column or table name
- ...

# Populating Tables

- `INSERT` statements
- Product specific load/copy statements
- GUI tools

# INSERT INTO by Constant Value

```
INSERT INTO <tableName> [(columnName [, columnName] ...)]
        VALUES (valueList) [, (valueList)]...;
```

# INSERT INTO by Constant Value Example

ROOM(classroomId, building, abbrev, room)

```
INSERT INTO Room (classroomId, building, abbrev, room) VALUES
   ('DCH1070','Duncan Hall', 'DCH', '1070'),
   ('DCH1055','Duncan Hall', 'DCH', '1055'),
   ('HRZ211','Herzstein Hall', 'HRZ', '211'),
   ('SEW305','Sewell Hall', 'SEW', '305');
```

# INSERT INTO by Constant Value Example

ROOM(classroomId, building, abbrev, room)

```
INSERT INTO Room (classroomId, building, room) VALUES
   ('DCH1070','Duncan Hall', '1070'),
   ('DCH1055','Duncan Hall', '1055'),
   ('HRZ211','Herzstein Hall',  '211'),
   ('SEW305','Sewell Hall',  '305');
```

? What goes into the abbrev field?

# INSERT INTO by Query

```
INSERT INTO <tableName> [(columnName [, columnName] ...)]
        <SELECT statement>;
```

# INSERT INTO by Query Example

MEMBER (memberId, lastName, firstName, memberType, startDate)
STUDENT(NETID, FIRSTNAME, LASTNAME)

```
INSERT INTO Member (memberId, lastname, firstName, memberType)
        SELECT netId, lastName, firstName, 'Student'
        FROM Student;
```

# INSERT INTO by Query Example 2

MEMBER (memberId, lastName, firstName, memberType, startDate)
STUDENT(NETID, FIRSTNAME, LASTNAME)

```
INSERT INTO Member (memberId, lastName, firstName, memberType)
        SELECT netId, lastName, firstName, 'Student'
        FROM Student
        WHERE netId NOT IN
                (SELECT memberId FROM Member);
```

? What does this query do?

# INSERT INTO Notes

- The entire query is evaluated before any tuples are inserted - this is why the statement on the last slide works
- The entire operation fails if just one value cannot be inserted
  - ? When might this happen?
- All constraints are checked

# Populating Tables in Bulk

- Every RDBMS has some variant of this functionality
  - `LOAD` in MySQL
  - `COPY` in postgres
- Faster and more space efficient than `INSERT INTO` commands
- You specify
  - File name and location (path)
  - Delimiter
  - Header / no header
  - ...

```
DELETE FROM <tableName>
   [WHERE where_clause]
```

- Deletes records from <tableName> that meet the specified criteria
- Reports how many rows were deleted
- Is slower than `TRUNCATE TABLE`
- Executes delete trigger(s)

# DELETE Example

ENROLL(NETID, CRN)

```
DELETE FROM Enroll
WHERE netId = 'abc1' AND crn = 12345;
```

| netId | crn |
|---|---|
| abc1 | 12345 |
| abc1 | 34567 |
| xyz5 | 12345 |
| hj9 | 89394 |

BECOMES

| netId | crn |
|---|---|
| | |
| abc1 | 34567 |
| xyz5 | 12345 |
| hj9 | 89394 |

# DELETE Implementation Details

Usually implemented in 2 passes

1. Marks all candidate rows that meet the WHERE condition
    1. Checks for constraints
    2. Entire contents of table is available
2. Deletes the marked rows
    1. Then updates indexes, etc.

# DELETE Based on Data in Another Table

COURSE(CRN, COURSENAME, SCHEDULE, STARTTIME, ENDTIME)
ENROLL(NETID, CRN)

```
DELETE
FROM Course
WHERE crn NOT IN
        (SELECT crn
         FROM Enroll);
```

? What does this query do?

# DELETE Based on Data in the Same Table

COURSE(CRN, COURSENAME, SCHEDULE, STARTTIME, ENDTIME)

```
DELETE
FROM Course
WHERE startTime =
        (SELECT MIN(startTime)
         FROM Course);
```

? What does this query do?

# DELETE (and UPDATE) Workaround

Usually implemented in 2 passes

1. Nest the query to force the creation of a temporary table

or

1. Put the keys in a temporary table
2. Then delete / update using the list of keys

```
UPDATE <tableName>
SET <set clause list>
[WHERE <search condition>]
```

- Updates the columns specified in the set clause list to the values specified

# UPDATE Example 1

STUDENT(NETID, FIRSTNAME, LASTNAME, GPA)

Reset the GPA of all students

```
UPDATE Student
   SET GPA = NULL;
```

# UPDATE Example 2

COURSE(CRN, COURSENAME, SCHEDULE, STARTTIME, ENDTIME)
STUDENT(NETID, FIRSTNAME, LASTNAME, GPA)
ENROLL(NETID, CRN)

? Set the GPA of all students enrolled in COMP 430 or COMP 533 to 4.0

# UPDATE Example 2

COURSE(CRN, COURSENAME, SCHEDULE, STARTTIME, ENDTIME)
STUDENT(NETID, FIRSTNAME, LASTNAME, GPA)
ENROLL(NETID, CRN)

- Set the GPA of all students enrolled in COMP 430 or COMP 533 to 4.0

```
UPDATE Student
        SET GPA = 4.0
        WHERE netId IN
                (SELECT netId
                 FROM Enroll e, Course C
                 WHERE e.crn = c.crn
                        AND courseName IN ('COMP␣533', 'COMP␣430'));
```

# UPDATE Example 3

STUDENT(NETID, FIRSTNAME, LASTNAME, GPA)
GRADE(NETID, CRN, ASSIGNMENT, GRADE)

? Set the GPA of all the students based on the average of their grades

```
UPDATE Student
SET gpa = (?);
```

# UPDATE Example 3

STUDENT(NETID, FIRSTNAME, LASTNAME, GPA)
GRADE(NETID, CRN, ASSIGNMENT, GRADE)

- Set the GPA of all the students based on the average of their grades

```
UPDATE Student
SET gpa = (SELECT ROUND(AVG(g.grade),2)
           FROM Grade g
           WHERE g.netId = student.netId
           GROUP BY g.netId);
```

# True/False Questions

1. If there is no WHERE clause in an UPDATE statement, no rows are updated
2. Data in CSV or tab delimited format CANNOT be loaded directly into a database
3. An UPDATE could be accomplished with a DELETE followed by an INSERT.
4. If we have the relation:
   STUDENT(NETID, FIRSTNAME, LASTNAME, GPA)
   Can we run:

```
INSERT INTO STUDENT(LASTNAME, NETID) VALUES ('Myers', 'rbm2')
```

   ?

# Schemas vs. Databases

- Used to describe the structure of a database
- Often used interchangeably
- Sometimes hierarchical
  - A database may contain many schemas
  - Each schema has its own namespace

# Multiple Schemas vs. Multiple Databases

- Think about permissions
- Think about size
- Think about backups
- Consult your Database Administrator!

# Wrap up

? How can we use what we learned today?

? What do we know now that we didn't know before?