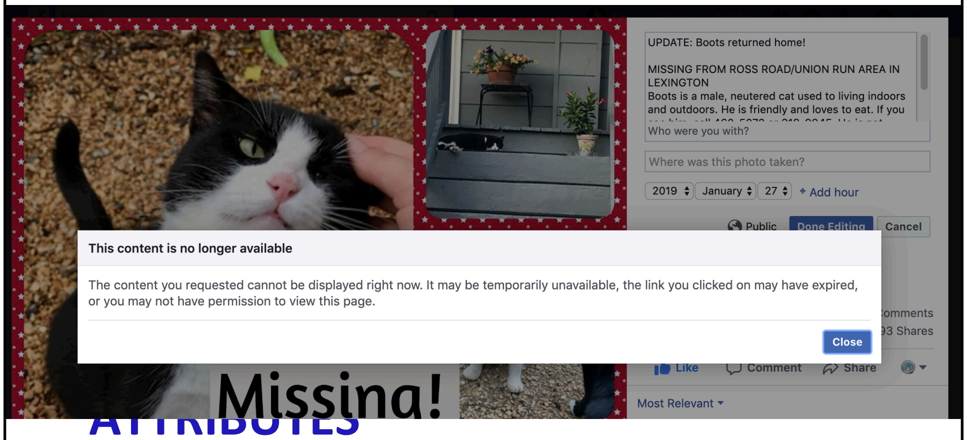
Objectives

- Review: JavaScript
- Quality Attributes of Web Software
- Introduction to Relational Databases, SQL
- JDBC

JavaScript review

- True or False: JavaScript is just like Java
- How do you declare a variable? (2 ways)
- How do you write text to the web page?
- What is the syntax for functions?
- What are some examples of events?
- How do you access a particular element in a document?
  - What are some ways to change that element?



UPDATE: Boots returned home!

MISSING FROM ROSS ROAD/UNION RUN AREA IN LEXINGTON
Boots is a male, neutered cat used to living indoors and outdoors. He is friendly and loves to eat. If you

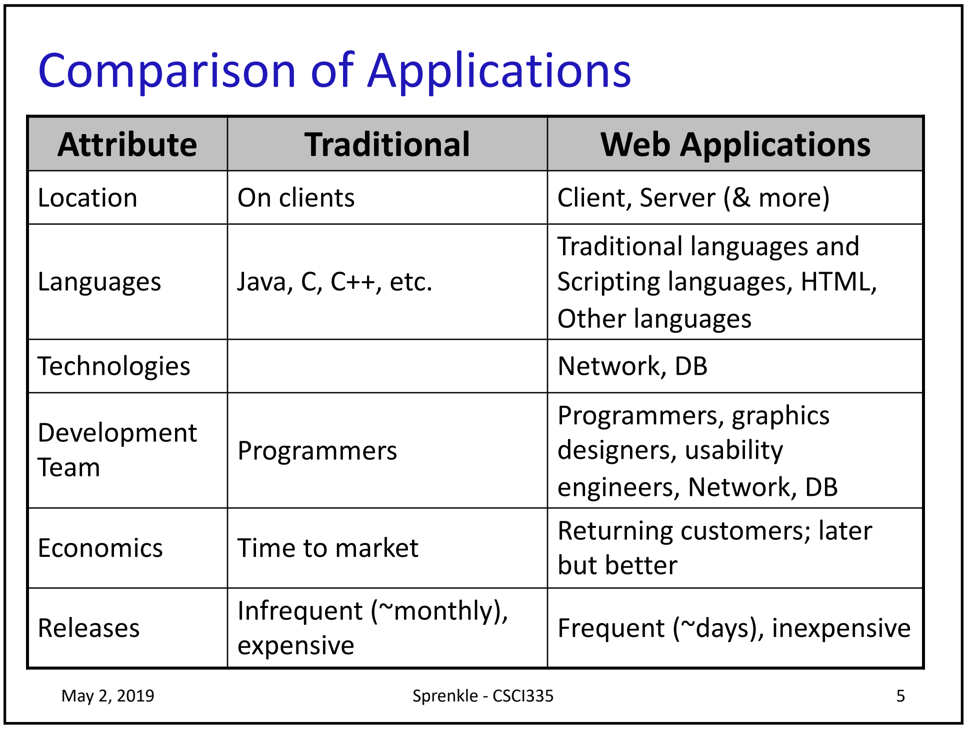
This content is no longer available

The content you requested cannot be displayed right now. It may be temporarily unavailable, the link you clicked on may have expired, or you may not have permission to view this page.

ATTRIBUTES

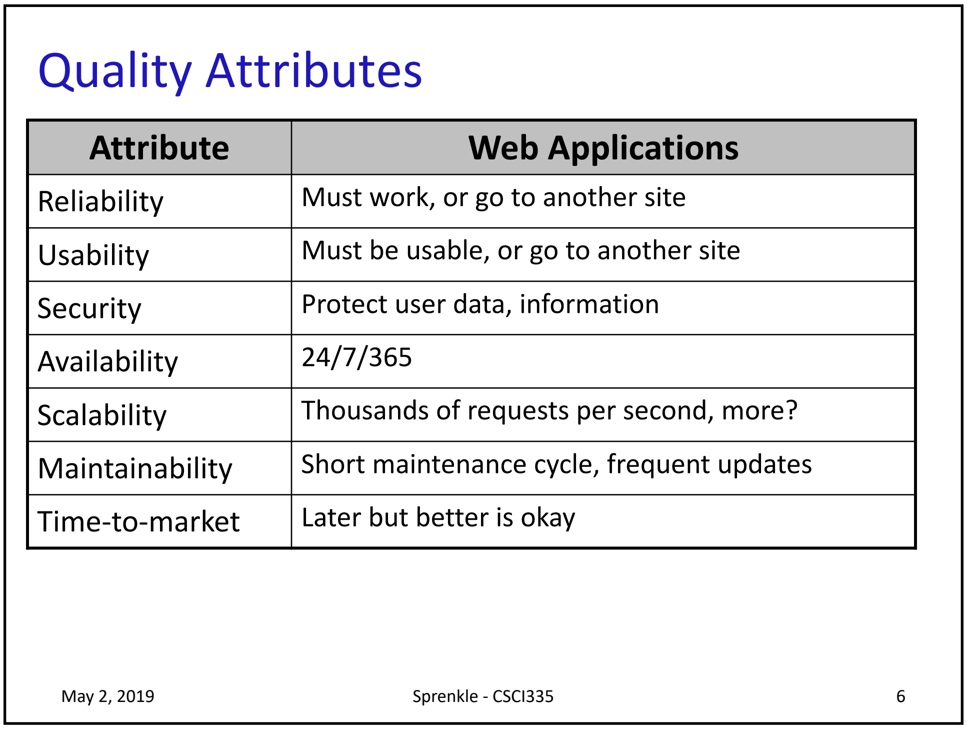
Quality Attributes

- How are web applications different from “traditional”/desktop applications?
  - Leads to differences in quality attributes
- React to “For most application types, commercial developers have traditionally had little motivation to produce high-quality software.”
- What are differences between 2002 (when article was originally published) and now?
- Let’s add another point in the comparison: mobile apps
  - Compare mobile apps with web and desktop



Comparison of Applications

Attribute	Traditional	Web Applications
Location	On clients	Client, Server (& more)
Languages	Java, C, C++, etc.	Traditional languages and Scripting languages, HTML, Other languages
Technologies		Network, DB
Development Team	Programmers	Programmers, graphics designers, usability engineers, Network, DB
Economics	Time to market	Returning customers; later but better
Releases	Infrequent (~monthly), expensive	Frequent (~days), inexpensive



Quality Attributes

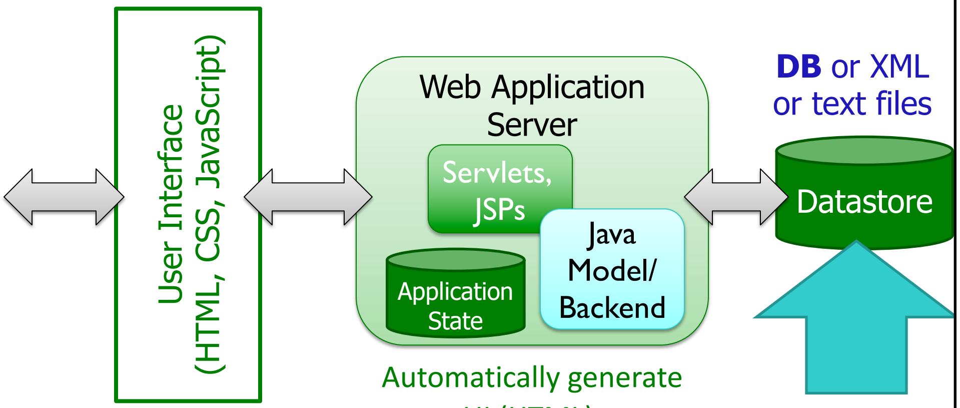
Attribute	Web Applications
Reliability	Must work, or go to another site
Usability	Must be usable, or go to another site
Security	Protect user data, information
Availability	24/7/365
Scalability	Thousands of requests per second, more?
Maintainability	Short maintenance cycle, frequent updates
Time-to-market	Later but better is okay

Discussion

- What are examples of sites that you used to use but you switched because something better came along?
  - How easy is it to switch now?

DATABASES AND SQL

Web Application Architecture Overview



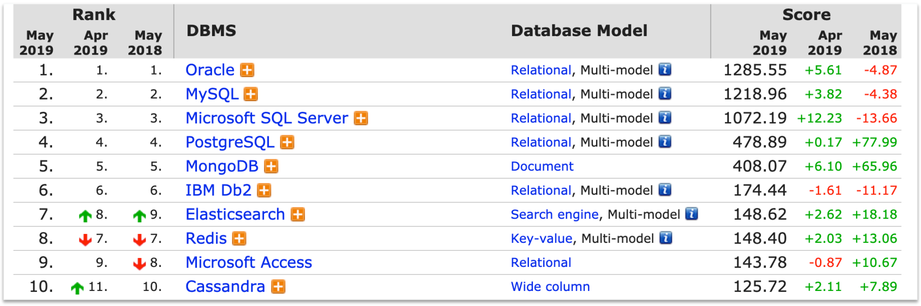
Automatically generate UI (HTML),
Handle user requests

Data structures,
helper classes

Database Overview

- Store data in such a way to allow *efficient* storage, search, and update
- **Relational Data Model** - currently most popular type of database
  - Many vendors: PostgreSQL, Oracle, MySQL, DB2, MSSQL
  - Data is stored in **tables**
  - ***Attributes***: column names (one word)
  - Often contain ***primary key***:
    a set of columns that uniquely identify a row

DB Popularity



Rank			DBMS	Database Model	Score		
May 2019	Apr 2019	May 2018			May 2019	Apr 2019	May 2018
1.	1.	1.	Oracle ⊞	Relational, Multi-model ℹ	1285.55	+5.61	-4.87
2.	2.	2.	MySQL ⊞	Relational, Multi-model ℹ	1218.96	+3.82	-4.38
3.	3.	3.	Microsoft SQL Server ⊞	Relational, Multi-model ℹ	1072.19	+12.23	-13.66
4.	4.	4.	PostgreSQL ⊞	Relational, Multi-model ℹ	478.89	+0.17	+77.99
5.	5.	5.	MongoDB ⊞	Document	408.07	+6.10	+65.96
6.	6.	6.	IBM Db2 ⊞	Relational, Multi-model ℹ	174.44	-1.61	-11.17
7.	↑ 8.	↑ 9.	Elasticsearch ⊞	Search engine, Multi-model ℹ	148.62	+2.62	+18.18
8.	↓ 7.	↓ 7.	Redis ⊞	Key-value, Multi-model ℹ	148.40	+2.03	+13.06
9.	9.	↓ 8.	Microsoft Access	Relational	143.78	-0.87	+10.67
10.	↑ 11.	10.	Cassandra ⊞	Wide column	125.72	+2.11	+7.89

Ranking based on web site mentions, searches, questions, job offers, professional profiles, social network mentions

https://db-engines.com/en/ranking

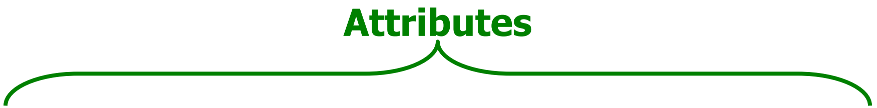
Example Students Table

- id is the *primary key*
- What are the attributes?

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2021	CSCI
43123	Brown	Allison	2020	ENGL

Example Students Table

- id is the primary key
- What are the attributes?



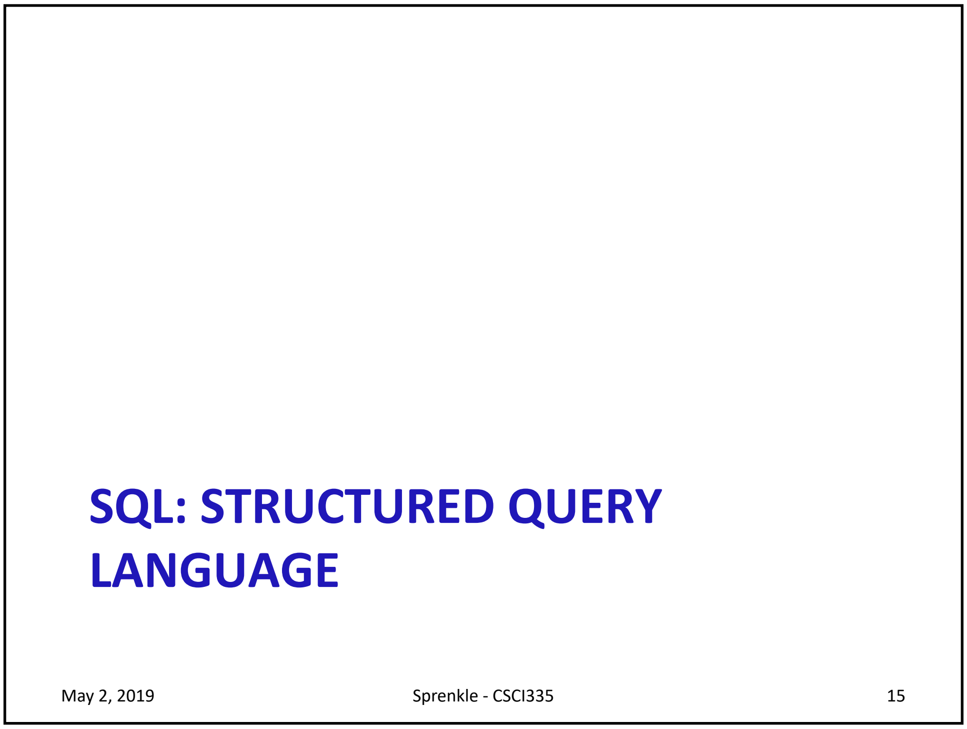
Attributes

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2021	CSCI
43123	Brown	Allison	2020	ENGL

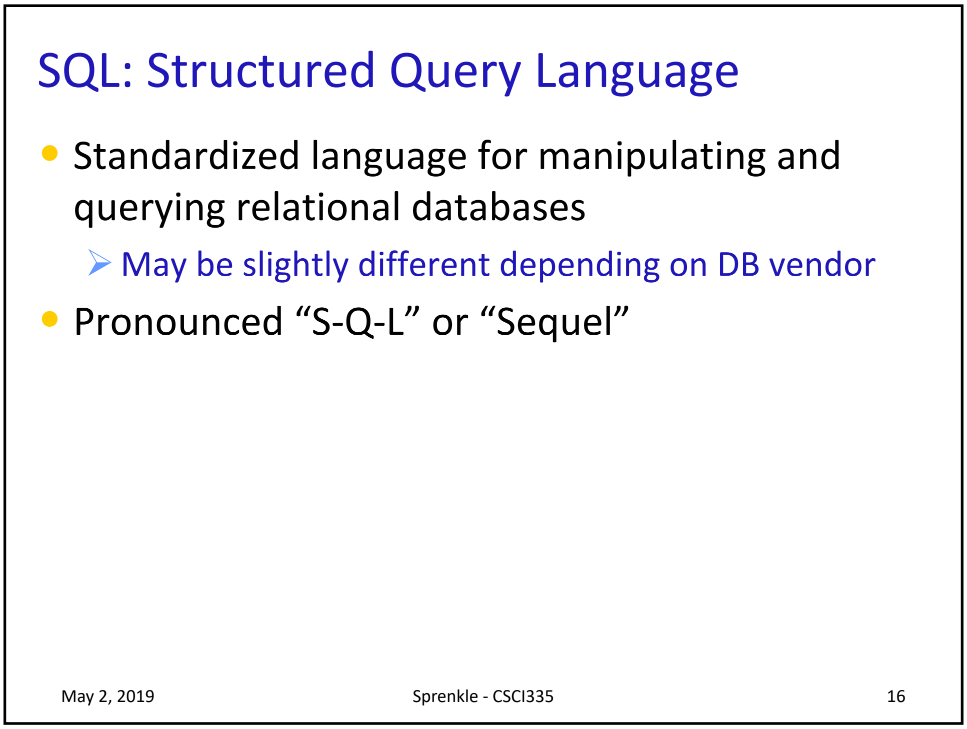
Courses Table

- Primary key is (Department, Number)
  - As a group, these uniquely identify a row

department	number	name	description
CSCI	101	Survey of Computer Science	A survey of …
CSCI	111	Fundamentals of Programming I	An introduction to …

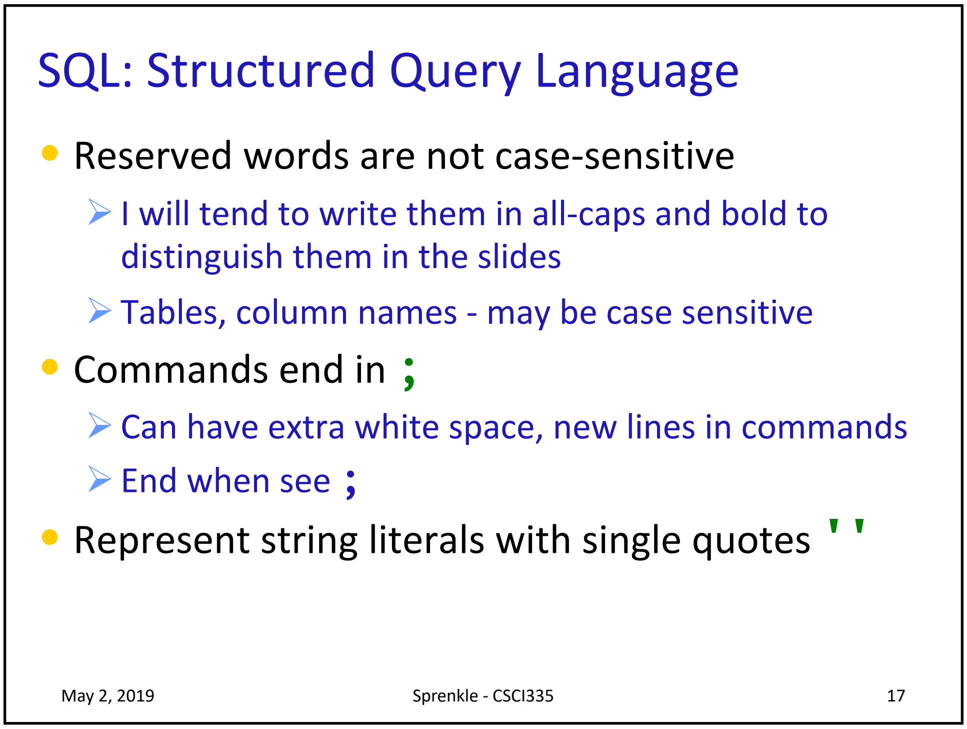


SQL: STRUCTURED QUERY LANGUAGE



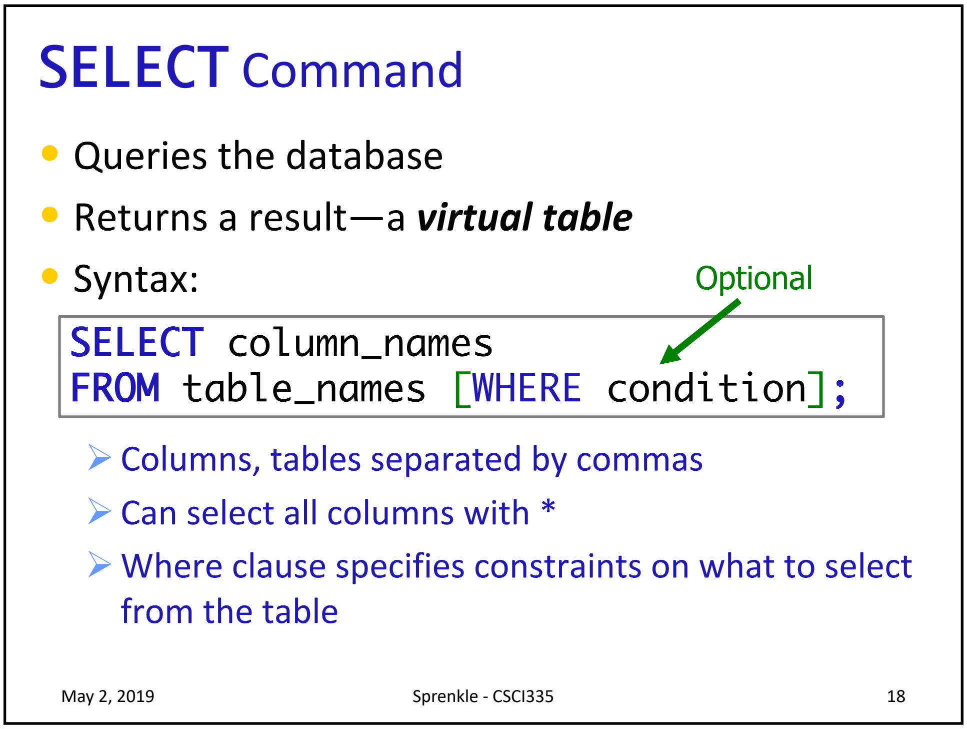
SQL: Structured Query Language

- Standardized language for manipulating and querying relational databases
  - May be slightly different depending on DB vendor
- Pronounced "S-Q-L" or "Sequel"



SQL: Structured Query Language

- Reserved words are not case-sensitive
  - I will tend to write them in all-caps and bold to distinguish them in the slides
  - Tables, column names - may be case sensitive
- Commands end in **;**
  - Can have extra white space, new lines in commands
  - End when see **;**
- Represent string literals with single quotes **' '**



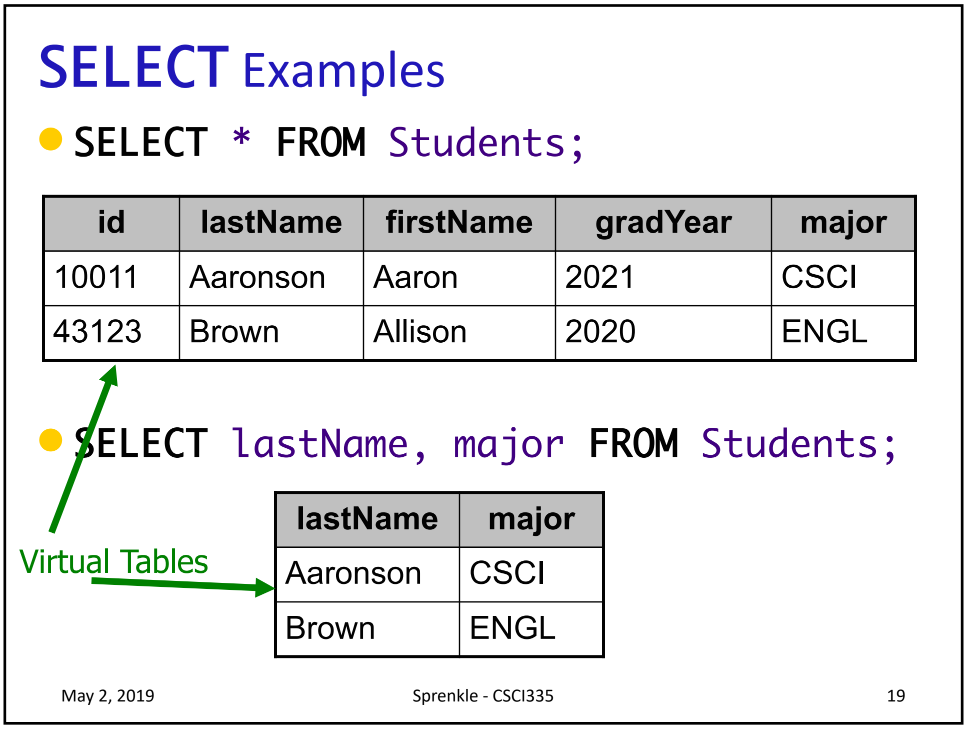
SELECT Command

- Queries the database
- Returns a result—a ***virtual table***
- Syntax:

Optional

```
SELECT column_names
FROM table_names [WHERE condition];
```

  - Columns, tables separated by commas
  - Can select all columns with *
  - Where clause specifies constraints on what to select from the table



SELECT Examples

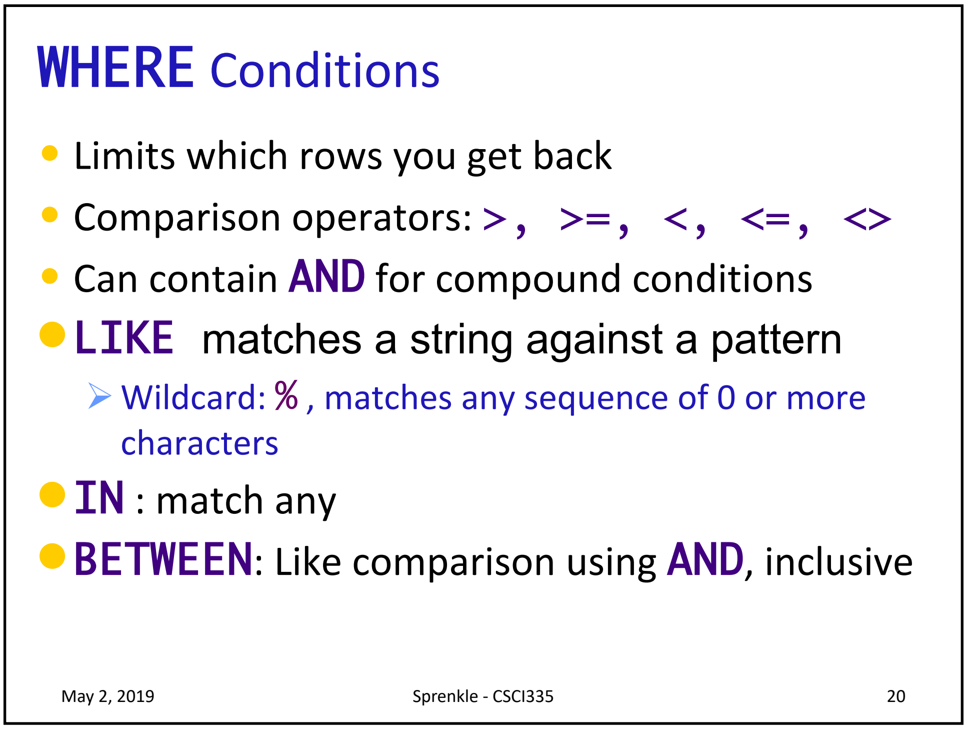
- SELECT * FROM Students;

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2021	CSCI
43123	Brown	Allison	2020	ENGL

- SELECT lastName, major FROM Students;

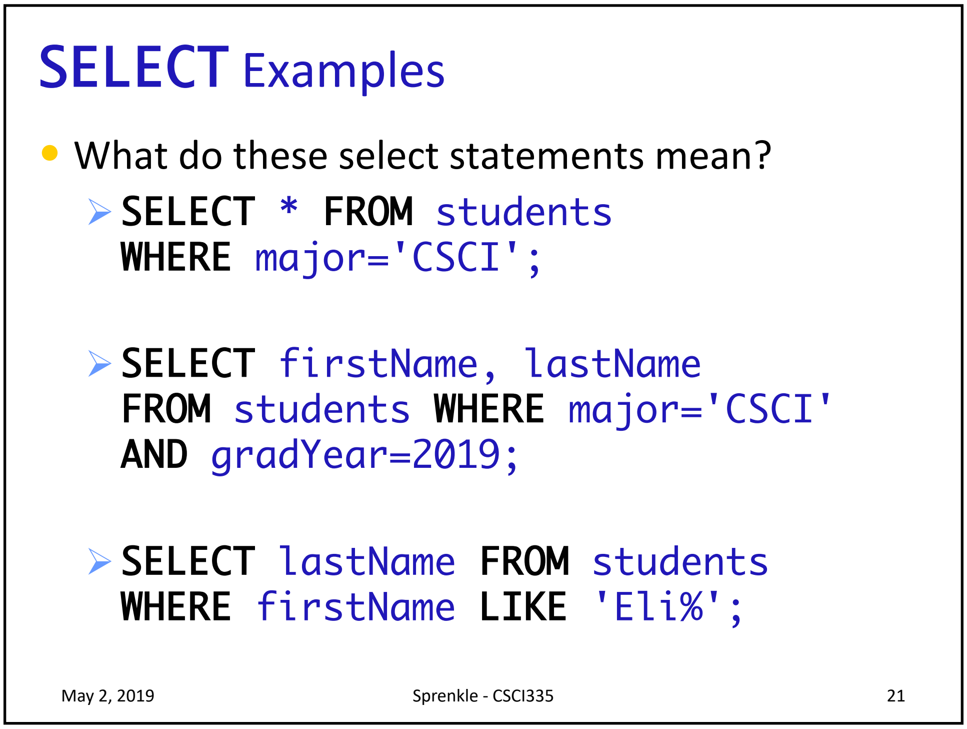
lastName	major
Aaronson	CSCI
Brown	ENGL

Virtual Tables



WHERE Conditions

- Limits which rows you get back
- Comparison operators: >, >=, <, <=, <>
- Can contain **AND** for compound conditions
- **LIKE** matches a string against a pattern
  - Wildcard: % , matches any sequence of 0 or more characters
- **IN** : match any
- **BETWEEN**: Like comparison using **AND**, inclusive



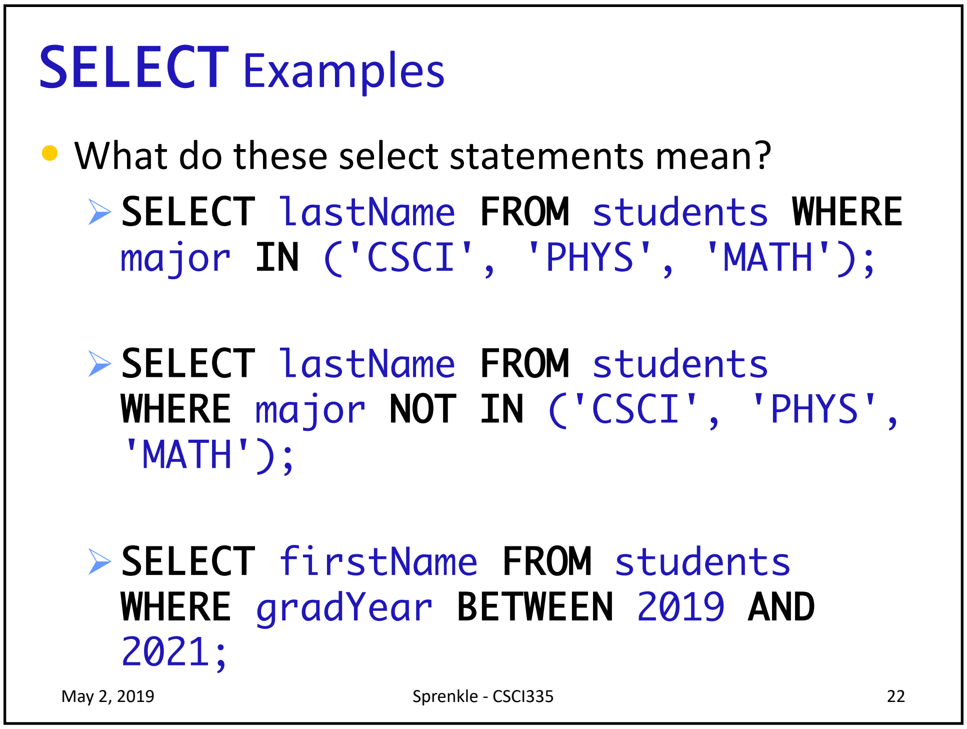
SELECT Examples

- What do these select statements mean?

  -
    ```
    SELECT * FROM students
    WHERE major='CSCI';
    ```

  -
    ```
    SELECT firstName, lastName
    FROM students WHERE major='CSCI'
    AND gradYear=2019;
    ```

  -
    ```
    SELECT lastName FROM students
    WHERE firstName LIKE 'Eli%';
    ```



SELECT Examples

- What do these select statements mean?

  -
    ```
    SELECT lastName FROM students WHERE
    major IN ('CSCI', 'PHYS', 'MATH');
    ```

  -
    ```
    SELECT lastName FROM students
    WHERE major NOT IN ('CSCI', 'PHYS',
    'MATH');
    ```

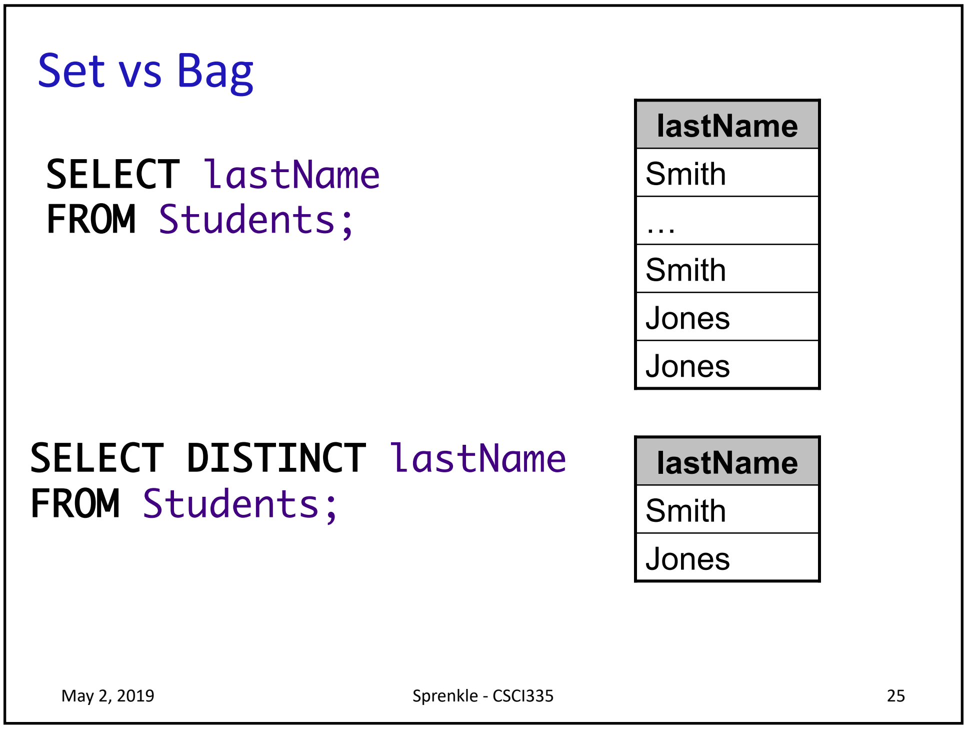
  -
    ```
    SELECT firstName FROM students
    WHERE gradYear BETWEEN 2019 AND
    2021;
    ```

Set vs Bag Semantics

- Data structures review

Set vs Bag Semantics

- Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - Used by SQL by default
- Set
  - No duplicates
  - Use keyword **`DISTINCT`**



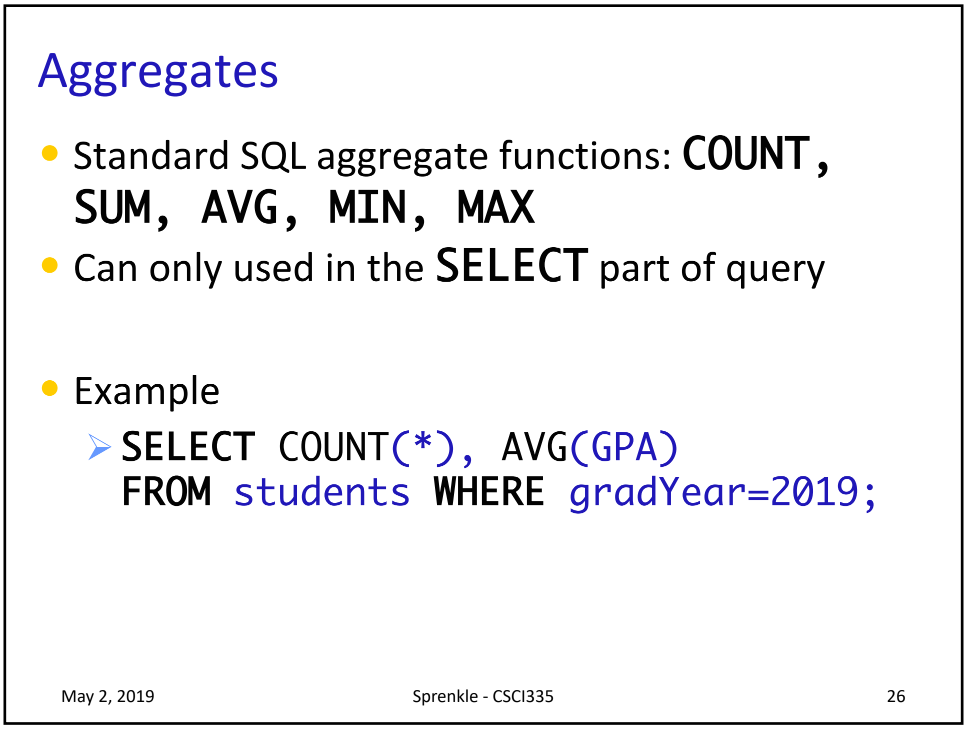
Set vs Bag

```
SELECT lastName
FROM Students;
```

lastName
Smith
…
Smith
Jones
Jones

```
SELECT DISTINCT lastName
FROM Students;
```

lastName
Smith
Jones



Aggregates

- Standard SQL aggregate functions: **COUNT, SUM, AVG, MIN, MAX**
- Can only used in the **SELECT** part of query
- Example

  -
    ```
    SELECT COUNT(*), AVG(GPA)
    FROM students WHERE gradYear=2019;
    ```

ORDER BY

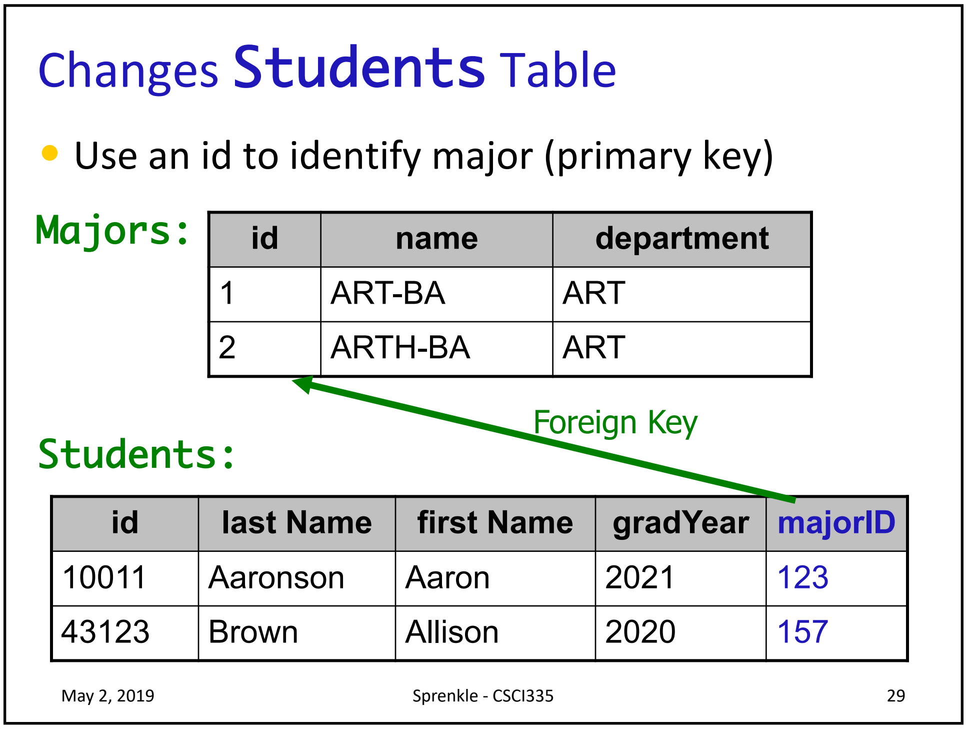
- Last operation performed, last in query
- Orders:
  - ASC = ascending
  - DESC = descending
- Example

  -
    ```
    SELECT firstName, lastName
    FROM Students WHERE gradYear=2019
    ORDER BY GPA DESC;
    ```

Majors Table

- Another table to keep track of majors
- Primary Key: id

id	name	department
1	ART-BA	ART
2	ARTH-BA	ART



Changes **Students** Table

- Use an id to identify major (primary key)

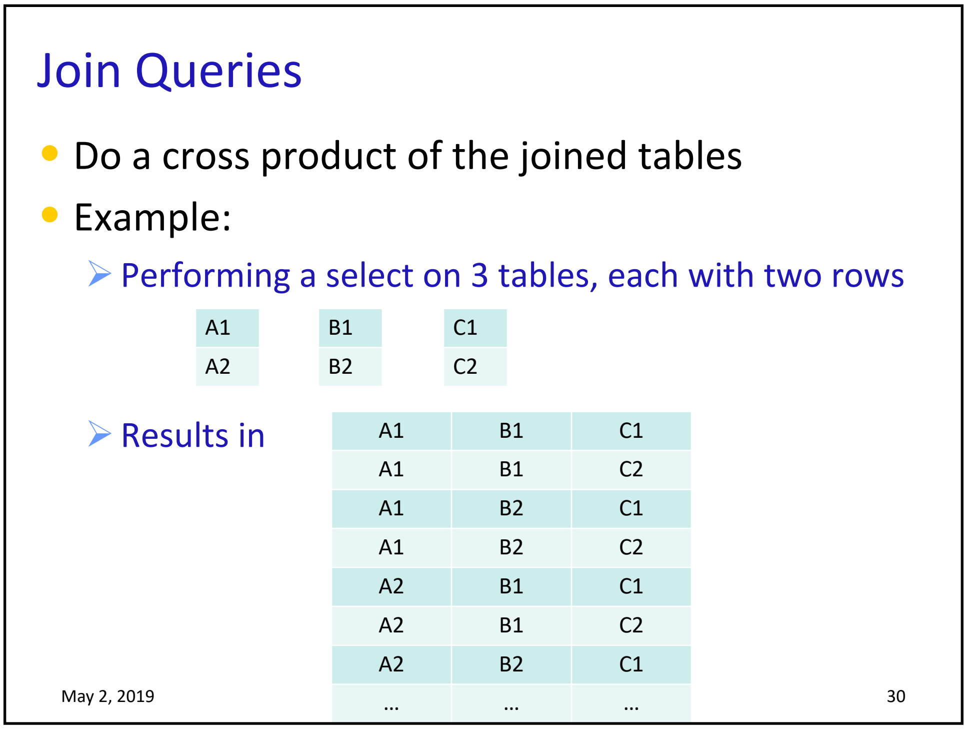
Majors:

id	name	department
1	ART-BA	ART
2	ARTH-BA	ART

Foreign Key

Students:

id	last Name	first Name	gradYear	majorID
10011	Aaronson	Aaron	2021	123
43123	Brown	Allison	2020	157



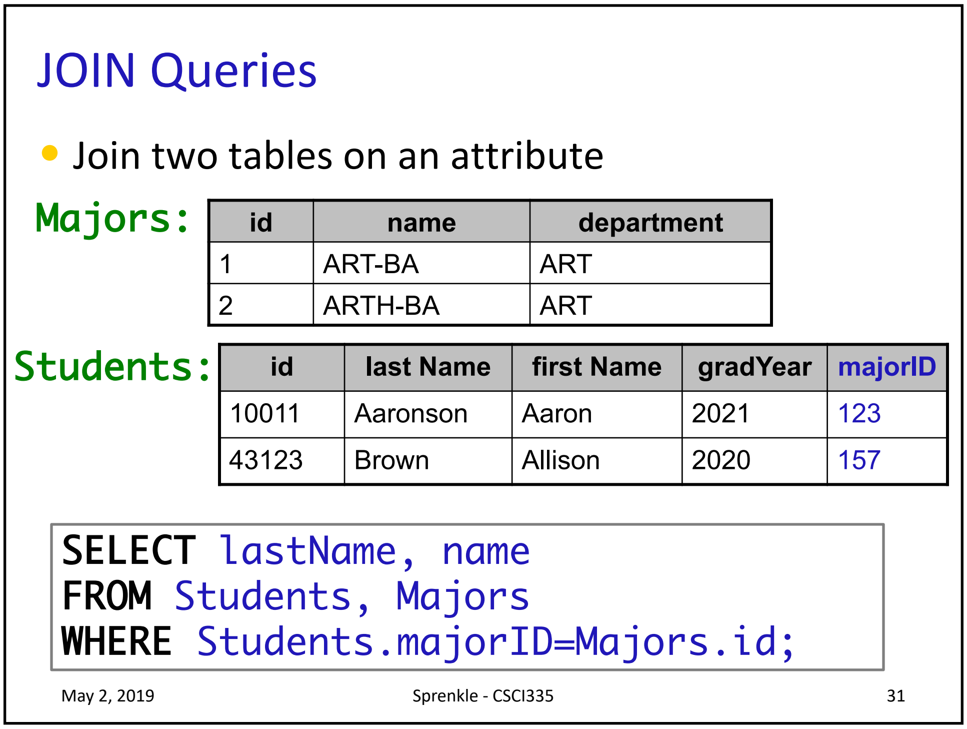
Join Queries

- Do a cross product of the joined tables
- Example:
  - Performing a select on 3 tables, each with two rows

A1	B1	C1
A2	B2	C2

  - Results in

A1	B1	C1
A1	B1	C2
A1	B2	C1
A1	B2	C2
A2	B1	C1
A2	B1	C2
A2	B2	C1
…	…	…



JOIN Queries

- Join two tables on an attribute

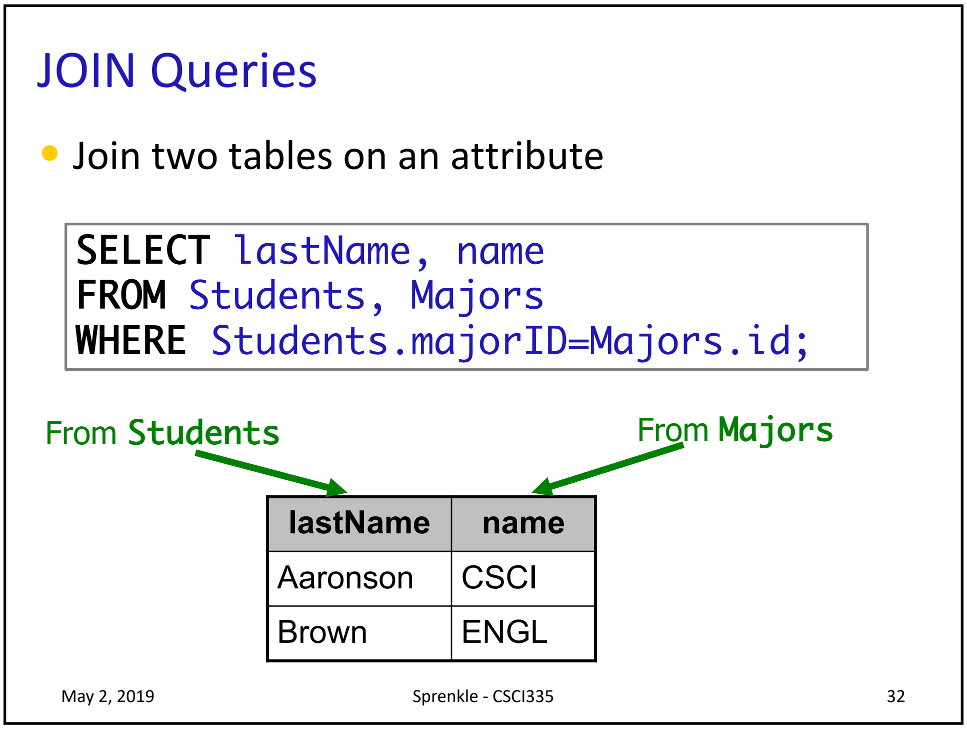
Majors:

id	name	department
1	ART-BA	ART
2	ARTH-BA	ART

Students:

id	last Name	first Name	gradYear	majorID
10011	Aaronson	Aaron	2021	123
43123	Brown	Allison	2020	157

```
SELECT lastName, name
FROM Students, Majors
WHERE Students.majorID=Majors.id;
```



JOIN Queries

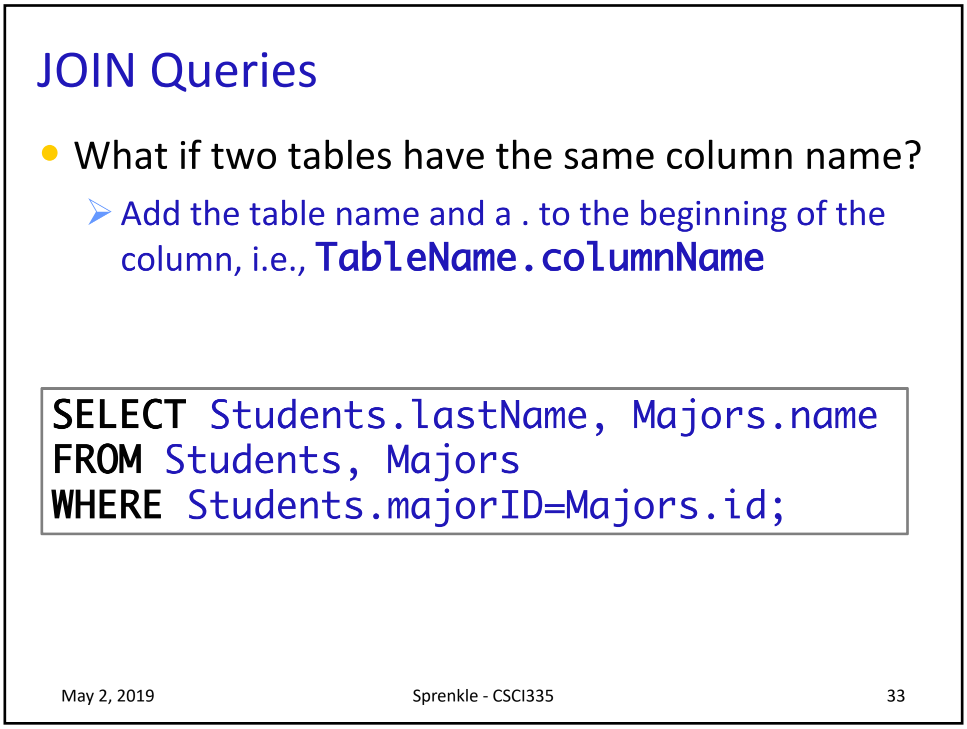
- Join two tables on an attribute

```
SELECT lastName, name
FROM Students, Majors
WHERE Students.majorID=Majors.id;
```

From Students

From Majors

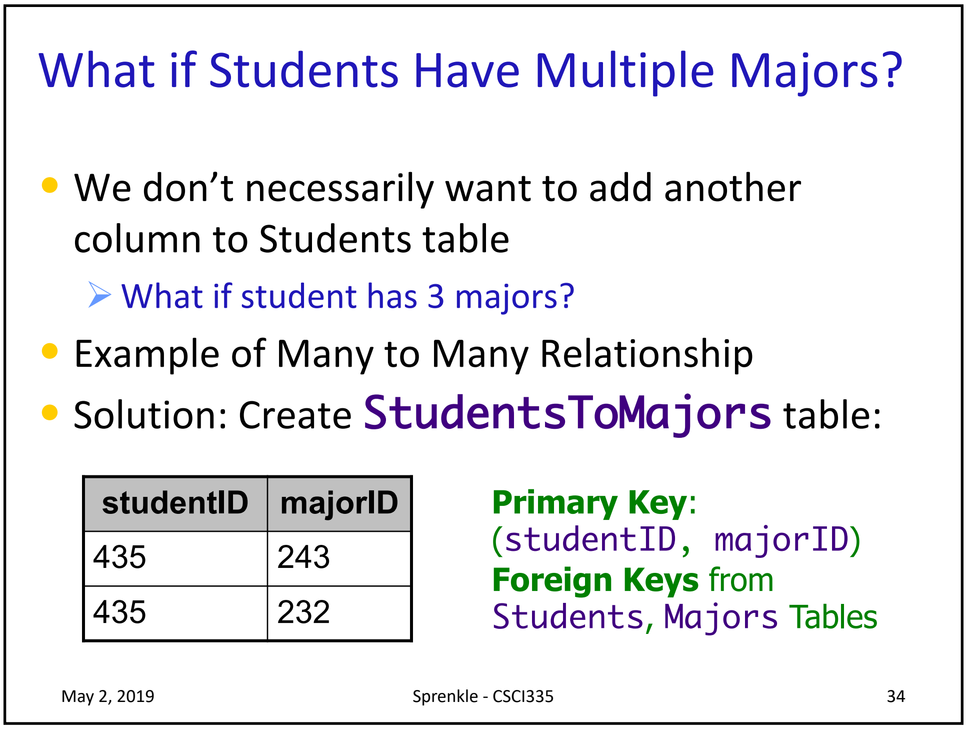
lastName	name
Aaronson	CSCI
Brown	ENGL



JOIN Queries

- What if two tables have the same column name?
  - Add the table name and a . to the beginning of the column, i.e., `TableName.columnName`

```
SELECT Students.lastName, Majors.name
FROM Students, Majors
WHERE Students.majorID=Majors.id;
```

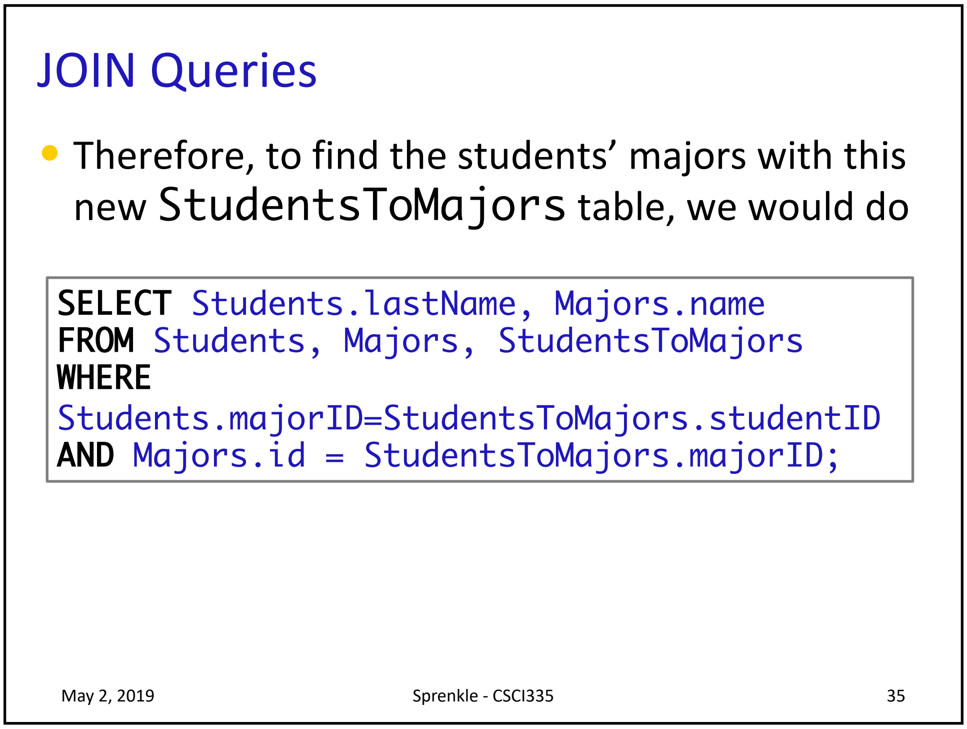


What if Students Have Multiple Majors?

- We don’t necessarily want to add another column to Students table
  - What if student has 3 majors?
- Example of Many to Many Relationship
- Solution: Create `StudentsToMajors` table:

studentID	majorID
435	243
435	232

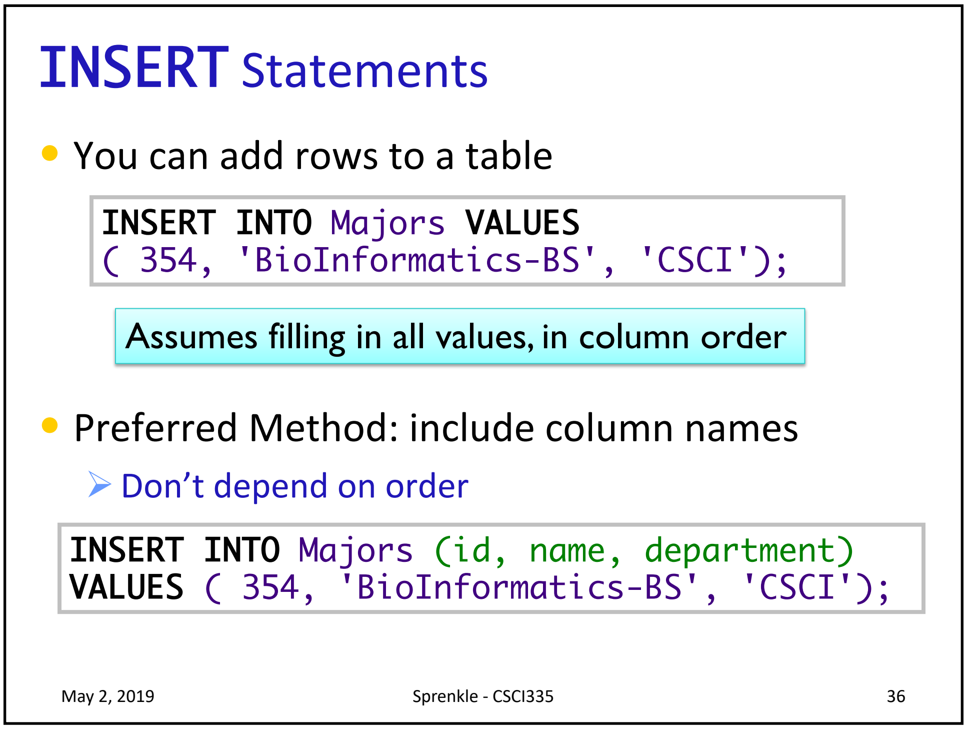
Primary Key: (`studentID, majorID`)
Foreign Keys from `Students`, `Majors` Tables



JOIN Queries

- Therefore, to find the students’ majors with this new StudentsToMajors table, we would do

```
SELECT Students.lastName, Majors.name
FROM Students, Majors, StudentsToMajors
WHERE
Students.majorID=StudentsToMajors.studentID
AND Majors.id = StudentsToMajors.majorID;
```



INSERT Statements

- You can add rows to a table

```
INSERT INTO Majors VALUES
( 354, 'BioInformatics-BS', 'CSCI');
```

Assumes filling in all values, in column order

- Preferred Method: include column names
  - Don’t depend on order

```
INSERT INTO Majors (id, name, department)
VALUES ( 354, 'BioInformatics-BS', 'CSCI');
```

INSERT Statements

- Automatically create ids

```
INSERT INTO Majors (id, name, department)
VALUES ( nextval('majors_sequence'),
'Bio-Informatics-BS', 'CSCI' );
```

- If table is set up appropriately, let the DB handle creating unique ids:

```
INSERT INTO Majors (name, department)
VALUES ( 'Bio-Informatics-BS', 'CSCI' );
```

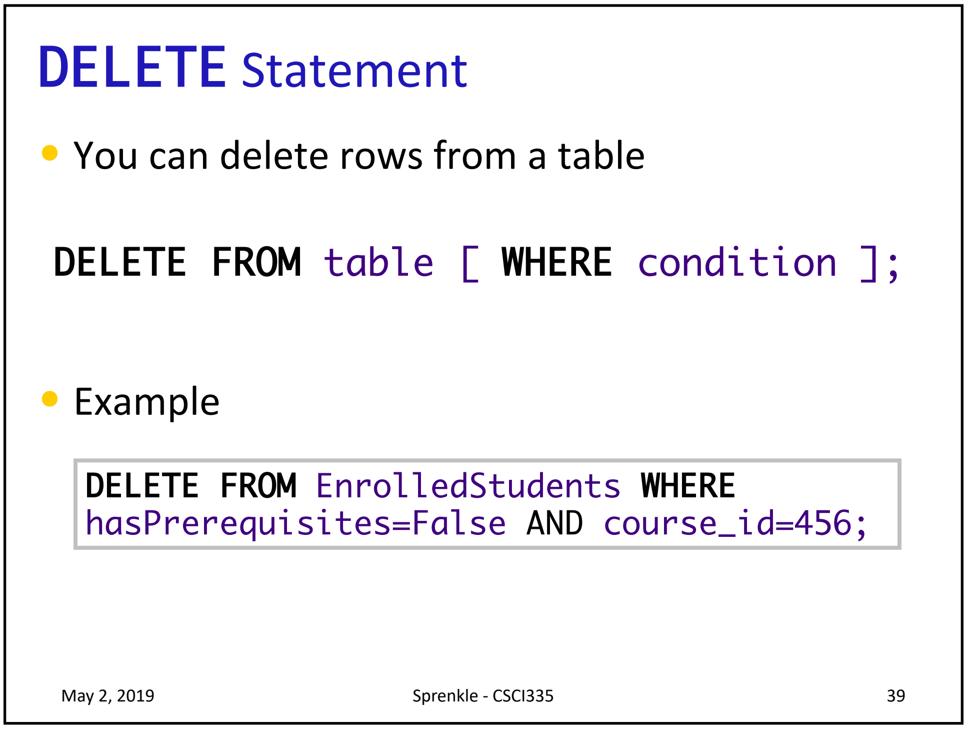
UPDATE Statement

- You can modify rows of a table
- Use **WHERE** condition to specify which rows to update
- Example: Update a student’s married name

```
UPDATE Students SET
LastName='Smith-Jones' WHERE id=12;
```

- Example: Update all first years to undeclared

```
UPDATE Students SET majorID=345
WHERE gradYear=2022;
```



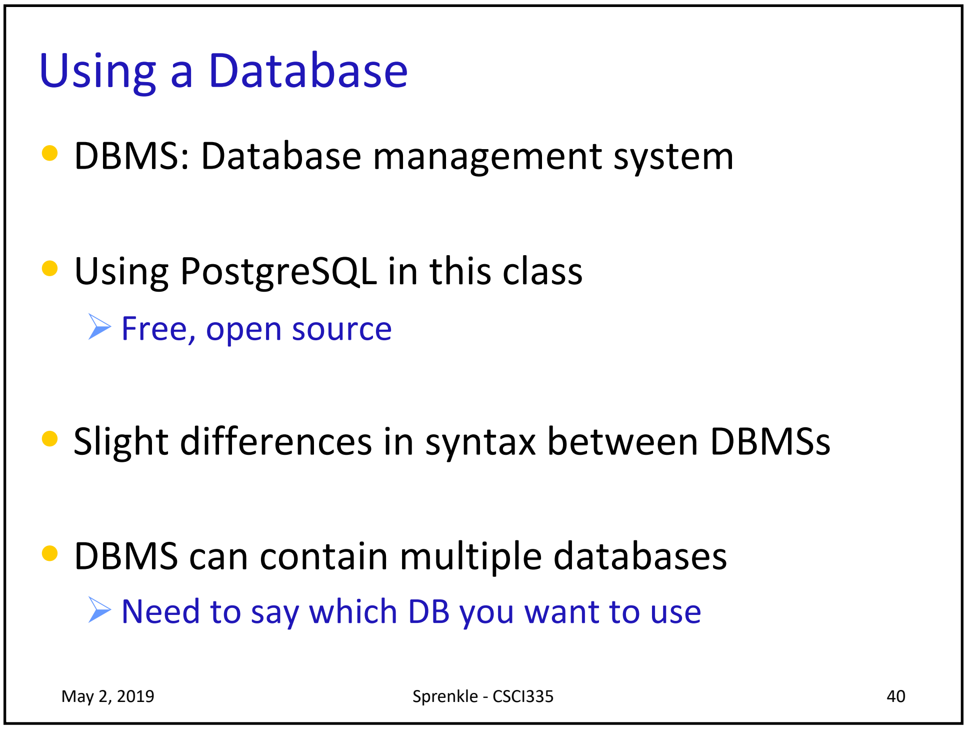
DELETE Statement

- You can delete rows from a table

```
DELETE FROM table [ WHERE condition ];
```

- Example

```
DELETE FROM EnrolledStudents WHERE
hasPrerequisites=False AND course_id=456;
```



Using a Database

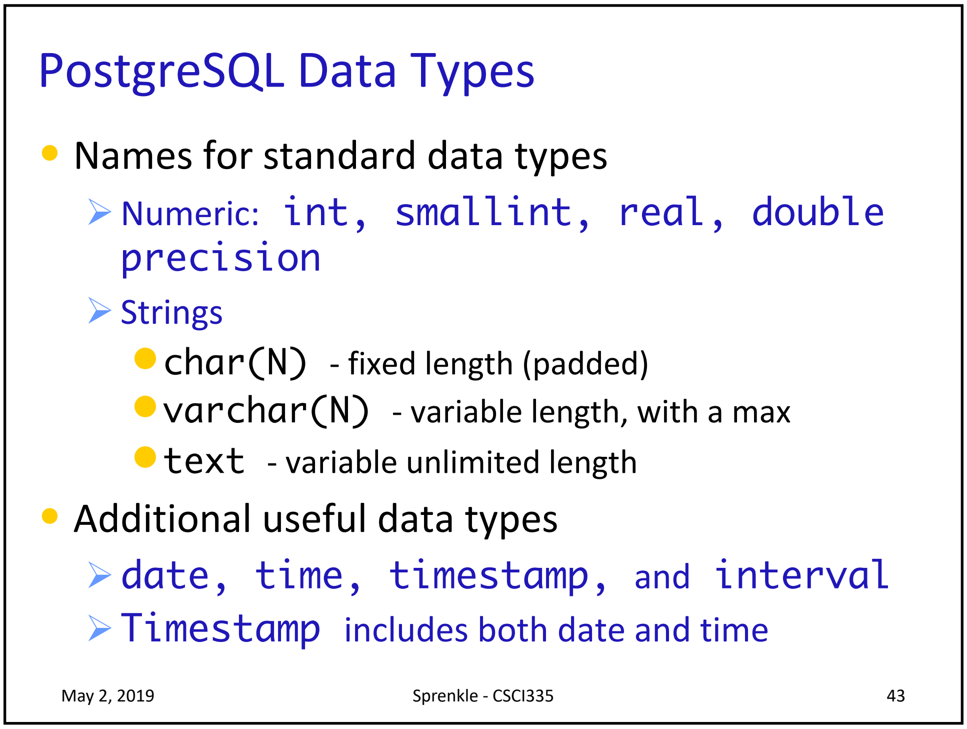
- DBMS: Database management system
- Using PostgreSQL in this class
  - Free, open source
- Slight differences in syntax between DBMSs
- DBMS can contain multiple databases
  - Need to say which DB you want to use

Designing a DB

- Design tables to hold your data
  - Data’s name and types
- Similar to OO design
  - No duplication of data
  - Have pointers to info in other tables
- Main difference: no lists
  - If you think “list”, think of a OneToMany or a ManyToMany table that contains the relationships between the data

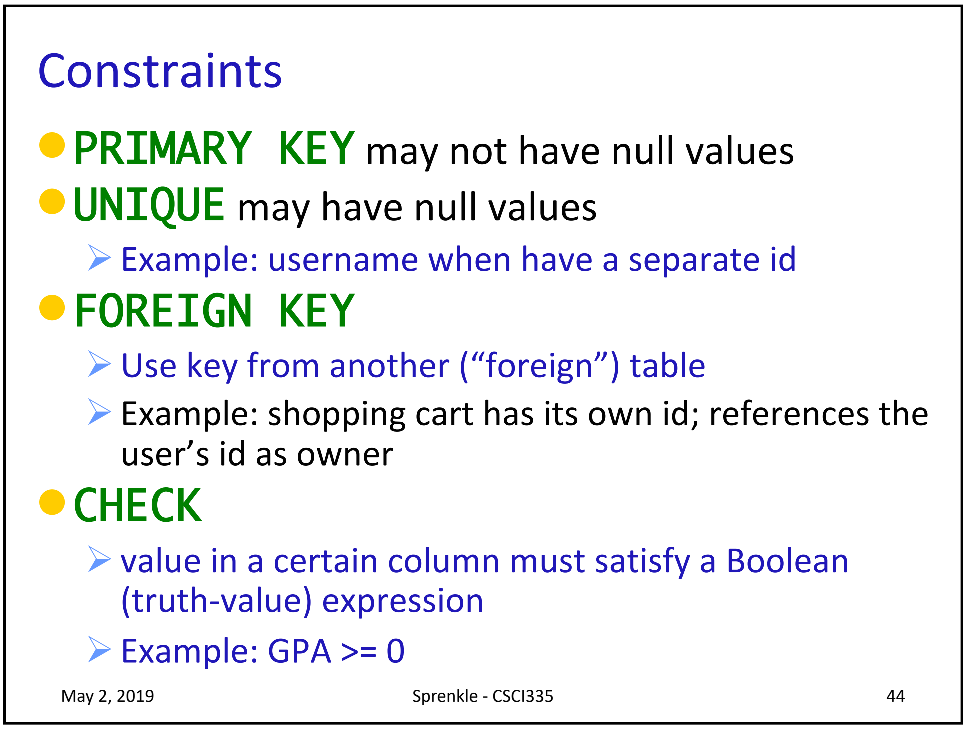
Standard Data Types

- Standard to SQL
  - CHAR - fixed-length character
  - VARCHAR - variable-length character
    - Requires more processing than CHAR
  - INTEGER - whole numbers
  - NUMERIC
  - Names for types in specific DB may vary
- More data types available in each DB



PostgreSQL Data Types

- Names for standard data types
  - Numeric: `int, smallint, real, double precision`
  - Strings
    - `char(N)` - fixed length (padded)
    - `varchar(N)` - variable length, with a max
    - `text` - variable unlimited length
- Additional useful data types
  - `date, time, timestamp,` and `interval`
  - `Timestamp` includes both date and time



Constraints

- **PRIMARY KEY** may not have null values
- **UNIQUE** may have null values
  - Example: username when have a separate id
- **FOREIGN KEY**
  - Use key from another (“foreign”) table
  - Example: shopping cart has its own id; references the user’s id as owner
- **CHECK**
  - value in a certain column must satisfy a Boolean (truth-value) expression
  - Example: GPA >= 0

Creating a Table

- Example:

```
CREATE TABLE weather (
    city            varchar(80),
    temp_lo         int,          -- low temperature
    temp_hi         int,          -- high temperature
    prcp            real,         -- precipitation
    date            date
);
```

Join Queries

- Joining two tables: creates a cross-product
- Where clauses restrict the number of results produced

“The Hack”

> Second Washington University hacked data base! Washington and Lee University full unedited database! gist.github.com/anonymous/4971… <https://t.co/3fqGJwXC>#SweetInfoOp <http://twitter.com/search?q=%23SweetInfoOp>

- Notified by W&L News Director
- President’s Day
- Actual link: https://gist.github.com/anonymous/4971936
  - Target : http://www.cs.wlu.edu/
  - Only some of the data, not all in database
- Figured out they just found my posted SQL file

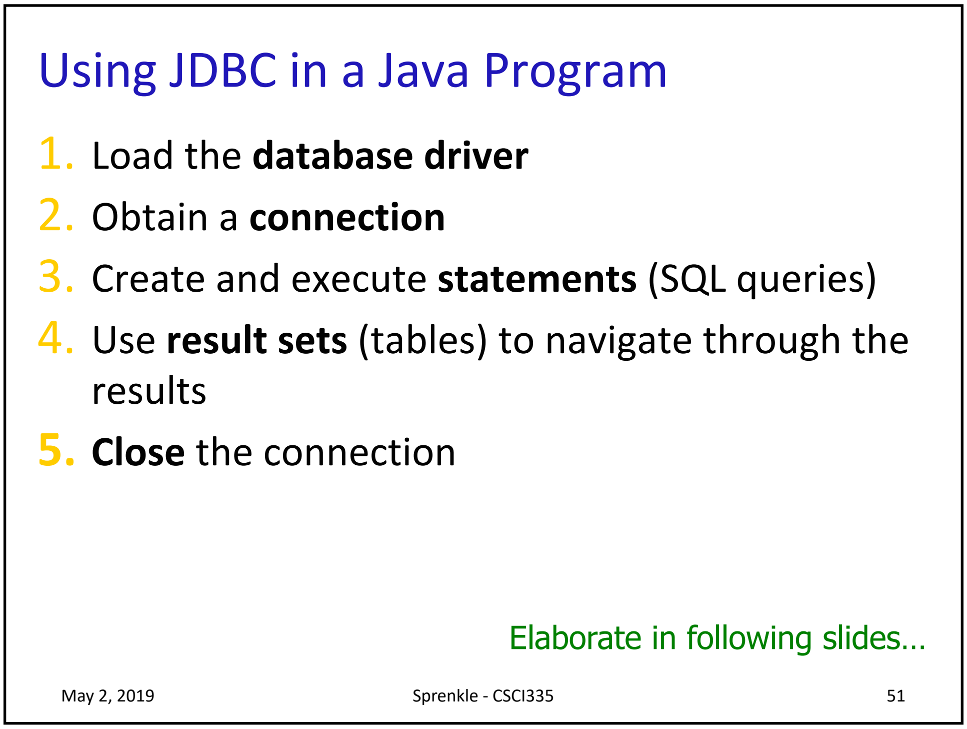
ChemTutor Database

- What tables will you need?
- What data?
- What constraints?

JDBC

JDBC: **J**ava **D**ata**b**ase **C**onnectivity

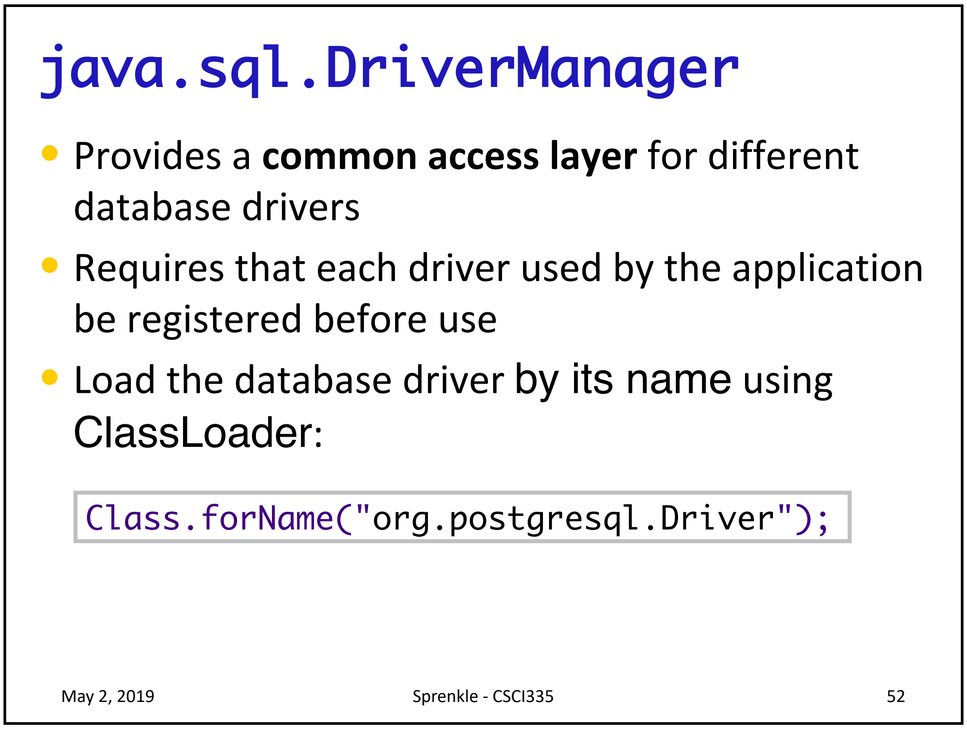
- Database-independent connectivity
    - JDBC converts generalized JDBC calls into vendor-specific SQL calls
- Classes in `java.sql.*` and `javax.sql.*` packages



Using JDBC in a Java Program

1. Load the **database driver**
2. Obtain a **connection**
3. Create and execute **statements** (SQL queries)
4. Use **result sets** (tables) to navigate through the results
5. **Close** the connection

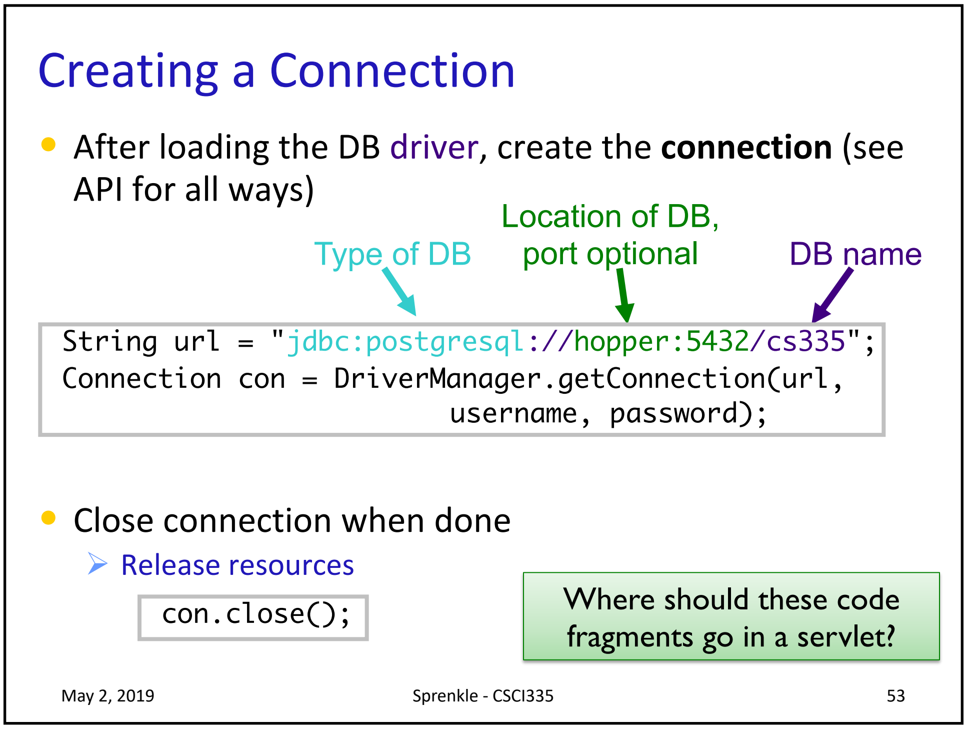
Elaborate in following slides…



java.sql.DriverManager

- Provides a **common access layer** for different database drivers
- Requires that each driver used by the application be registered before use
- Load the database driver by its name using ClassLoader:

```
Class.forName("org.postgresql.Driver");
```



Creating a Connection

- After loading the DB driver, create the **connection** (see API for all ways)

Type of DB

Location of DB, port optional

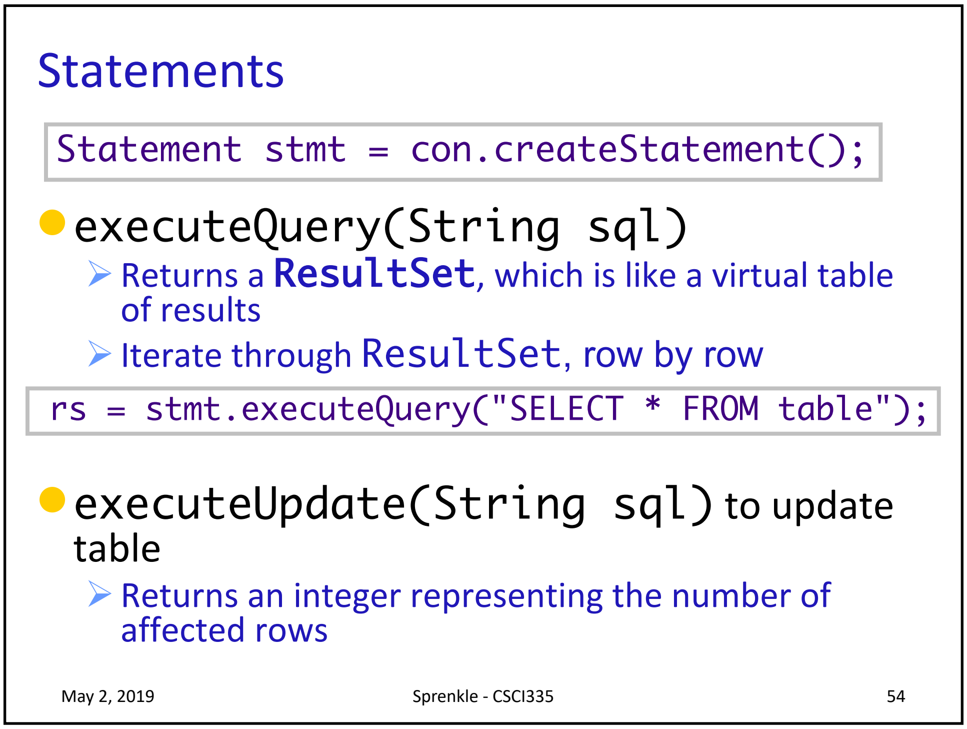
DB name

```
String url = "jdbc:postgresql://hopper:5432/cs335";
Connection con = DriverManager.getConnection(url,
                        username, password);
```

- Close connection when done
  - Release resources

```
con.close();
```

Where should these code fragments go in a servlet?



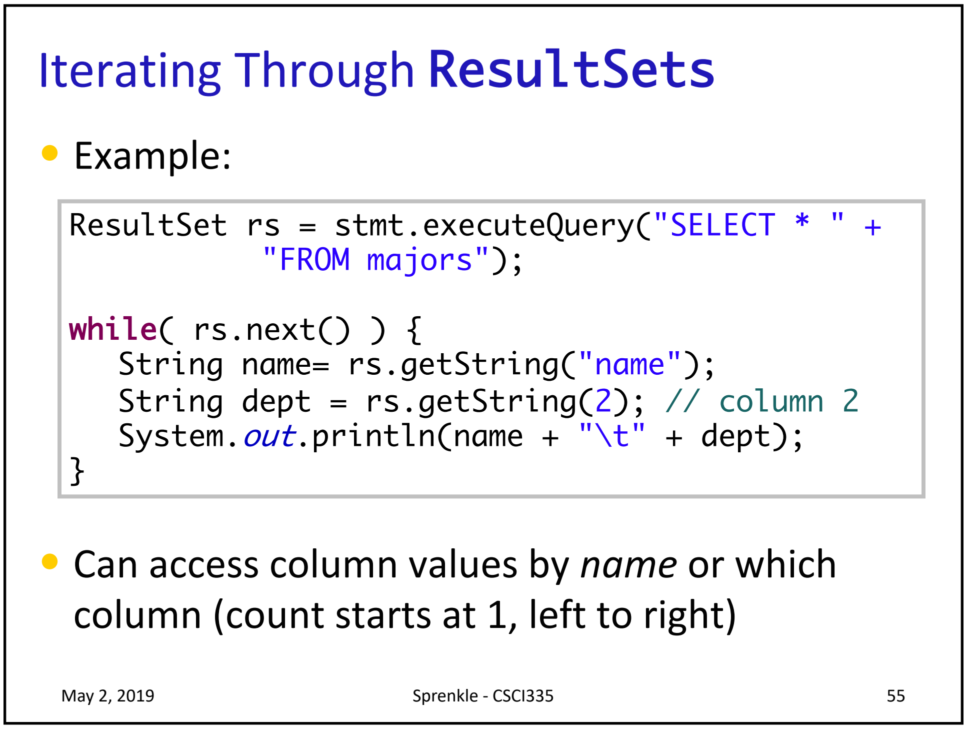
Statements

```
Statement stmt = con.createStatement();
```

- executeQuery(String sql)
  - Returns a **ResultSet**, which is like a virtual table of results
  - Iterate through ResultSet, row by row

```
rs = stmt.executeQuery("SELECT * FROM table");
```

- executeUpdate(String sql) to update table
  - Returns an integer representing the number of affected rows



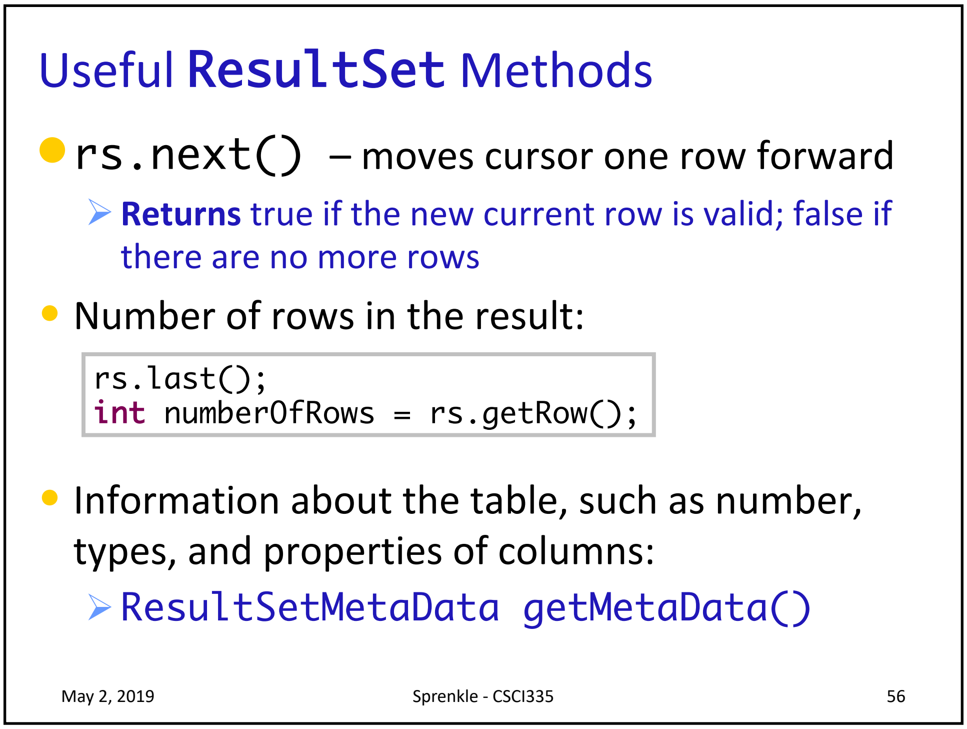
Iterating Through ResultSets

- Example:

```
ResultSet rs = stmt.executeQuery("SELECT * " +
           "FROM majors");

while( rs.next() ) {
   String name= rs.getString("name");
   String dept = rs.getString(2); // column 2
   System.out.println(name + "\t" + dept);
}
```

- Can access column values by *name* or which column (count starts at 1, left to right)

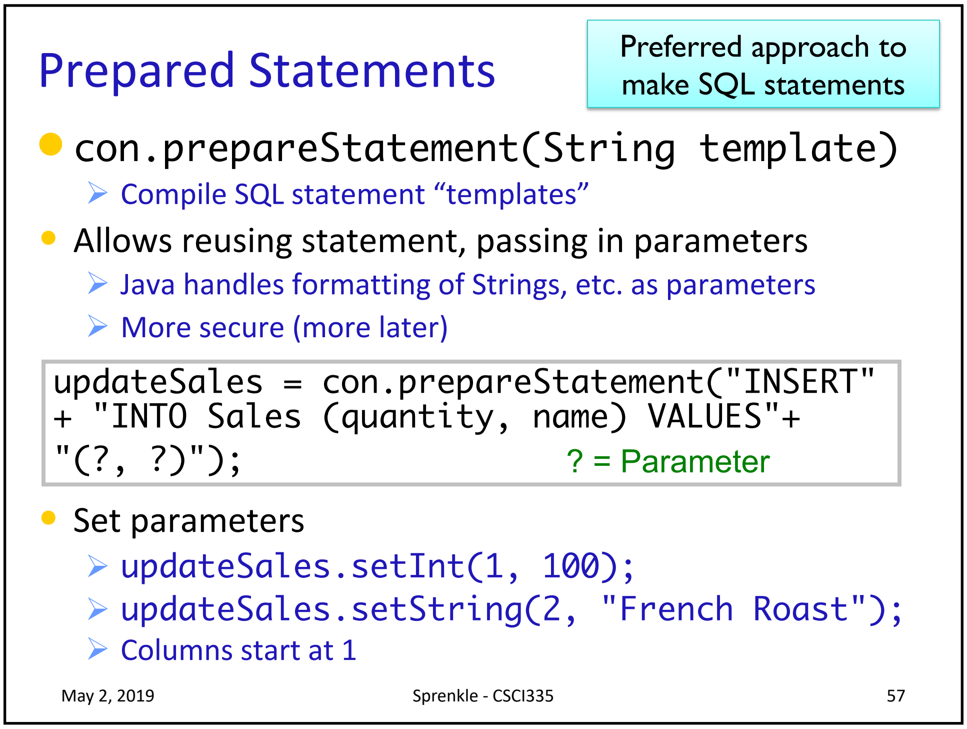


Useful ResultSet Methods

- `rs.next()` – moves cursor one row forward
  - **Returns** true if the new current row is valid; false if there are no more rows
- Number of rows in the result:

```
rs.last();
int numberOfRows = rs.getRow();
```

- Information about the table, such as number, types, and properties of columns:
  - `ResultSetMetaData getMetaData()`



Prepared Statements

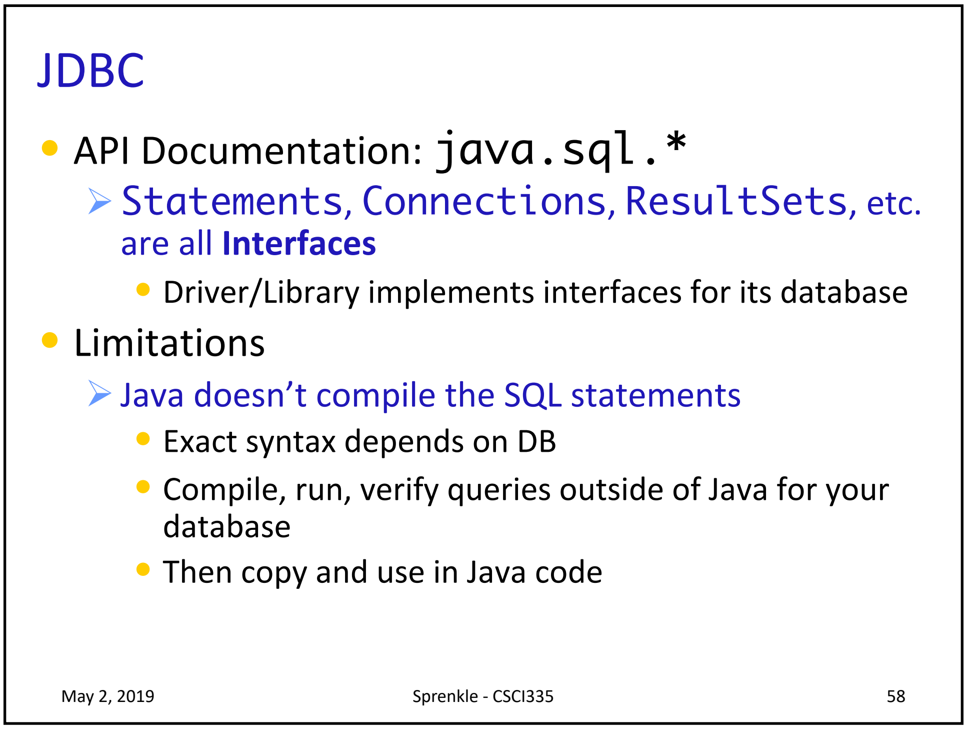
Preferred approach to make SQL statements

- `con.prepareStatement(String template)`
  - Compile SQL statement “templates”
- Allows reusing statement, passing in parameters
  - Java handles formatting of Strings, etc. as parameters
  - More secure (more later)

```
updateSales = con.prepareStatement("INSERT"
+ "INTO Sales (quantity, name) VALUES"+
"(?, ?)");
```

? = Parameter

- Set parameters
  - `updateSales.setInt(1, 100);`
  - `updateSales.setString(2, "French Roast");`
  - Columns start at 1



JDBC

- API Documentation: `java.sql.*`
  - `Statements`, `Connections`, `ResultSets`, etc. are all **Interfaces**
    - Driver/Library implements interfaces for its database
- Limitations
  - Java doesn’t compile the SQL statements
    - Exact syntax depends on DB
    - Compile, run, verify queries outside of Java for your database
    - Then copy and use in Java code

Using PostgreSQL on Command-Line

- In a terminal, **ssh** into **hopper**
  - ssh -XY hopper
- Run the PostgreSQL client: **psql** , connecting to the appropriate database
  - psql cs335
- At the prompt, type in SQL statements, ending in ;

Examples Using JDBC

Transactions in JDBC

- By default, a connection is in **auto-commit** mode
  - Each statement is a transaction
  - Automatically committed as soon as executed

Transactions in JDBC

- You can turn off auto-commit and execute multiple statements as a transaction
  - Database can keep handling others' reads
  - Others won't see updates until you commit

```
con.setAutoCommit(false);
// execute SQL statements …
con.commit(); // commit those statements
con.setAutoCommit(true);
```

- Can call `rollback` to abort updates

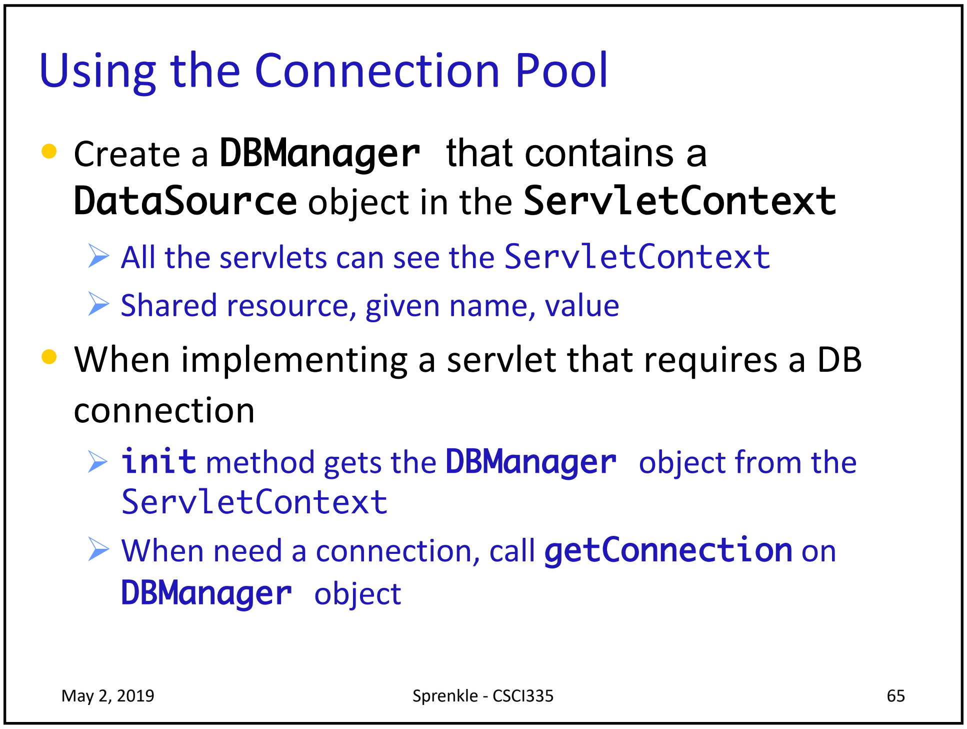
Storing Passwords

- Use **md5** function on passwords
  - `md5('password')`
- Compare user’s input password md5’d with password in database
  - `SELECT COUNT(id) FROM Users WHERE username=? AND password=md5(?);`
  - What are the possible outputs from this query?
- Example: username and password = ‘test’

There are stronger ways to encrypt passwords, but for this practice exercise, this is fine.

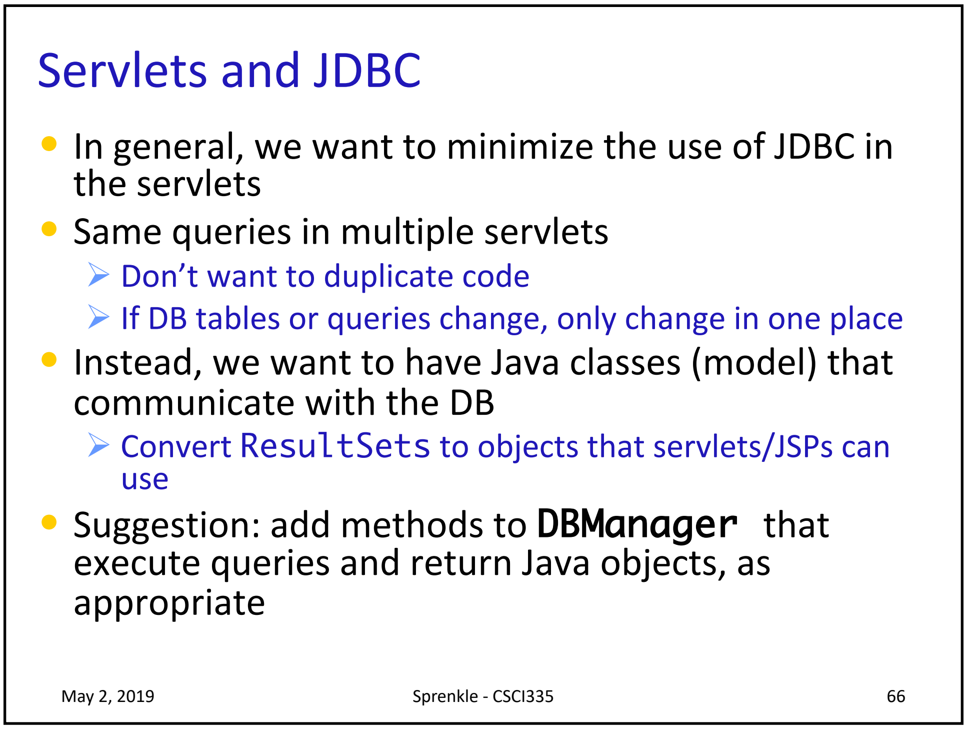
Connection Pool

- Want to reuse DB connections
  - Reduce overhead of creating and closing connections to database
- Could write our own connection pool class
  - Many examples online
- Apache wrote the one that we’ll use
  - `http://commons.apache.org/dbcp/`



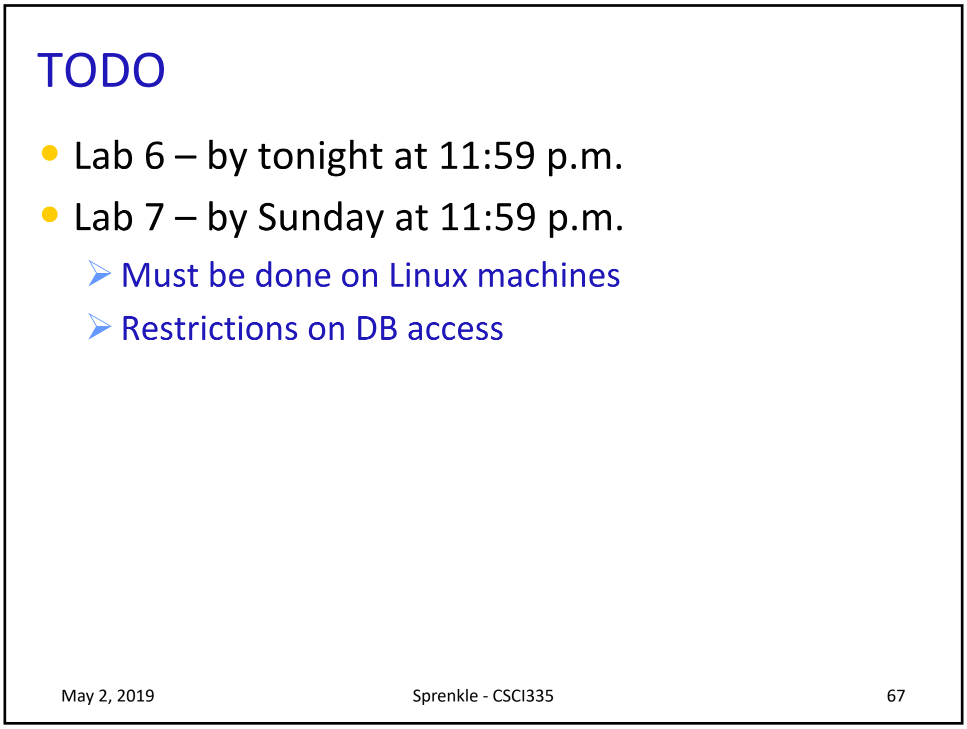
Using the Connection Pool

- Create a **`DBManager`** that contains a **`DataSource`** object in the **`ServletContext`**
  - All the servlets can see the `ServletContext`
  - Shared resource, given name, value
- When implementing a servlet that requires a DB connection
  - **`init`** method gets the **`DBManager`** object from the `ServletContext`
  - When need a connection, call **`getConnection`** on **`DBManager`** object



Servlets and JDBC

- In general, we want to minimize the use of JDBC in the servlets
- Same queries in multiple servlets
  - Don’t want to duplicate code
  - If DB tables or queries change, only change in one place
- Instead, we want to have Java classes (model) that communicate with the DB
  - Convert `ResultSets` to objects that servlets/JSPs can use
- Suggestion: add methods to **`DBManager`** that execute queries and return Java objects, as appropriate



TODO

- Lab 6 – by tonight at 11:59 p.m.
- Lab 7 – by Sunday at 11:59 p.m.
  - Must be done on Linux machines
  - Restrictions on DB access