Objectives

- Backend: Data stores

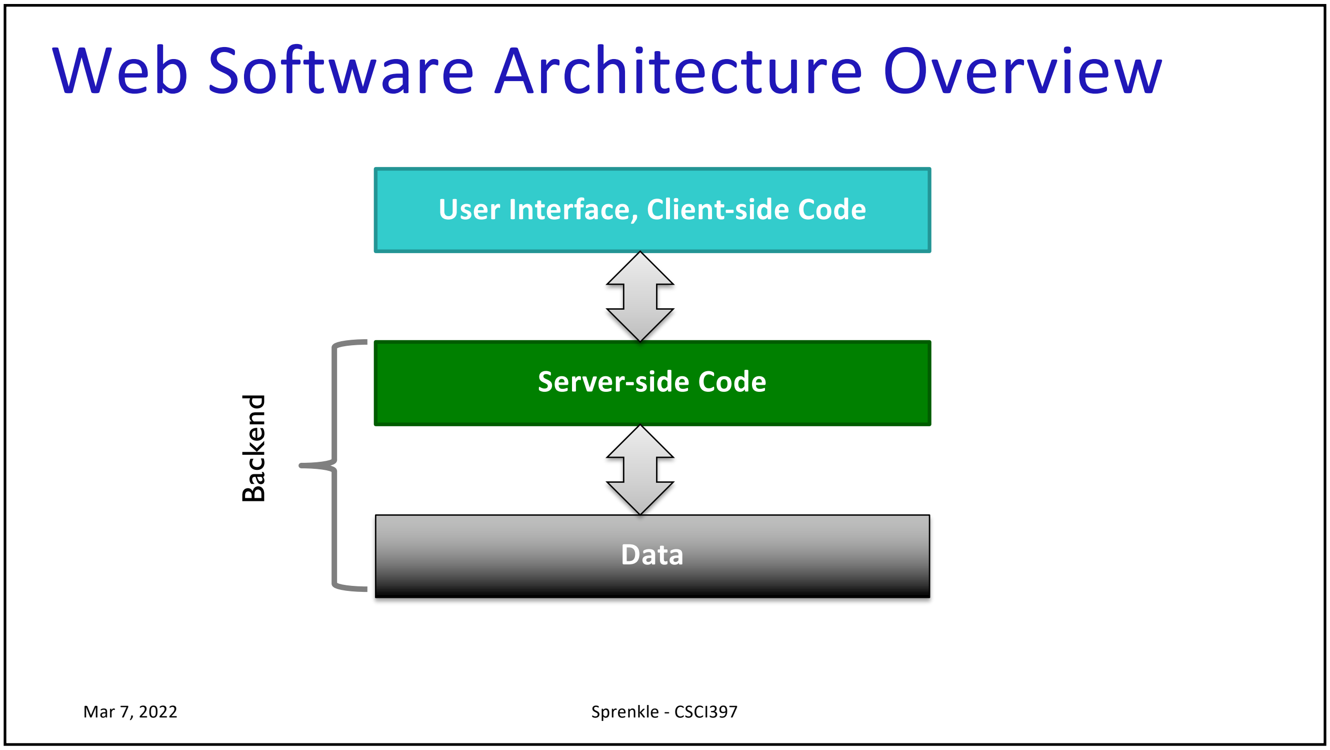
Review: Software Engineering at Google

- How is software engineering at Google different than software engineering in academia?

Review: Google's Concerns

- May boil down to size, scale, and time
- Availability
- Change over time
- How many bugs can be in your release?
- How many features should you support?

FULL-STACK DEVELOPMENT: DATA STORAGE



Web Software Architecture Overview

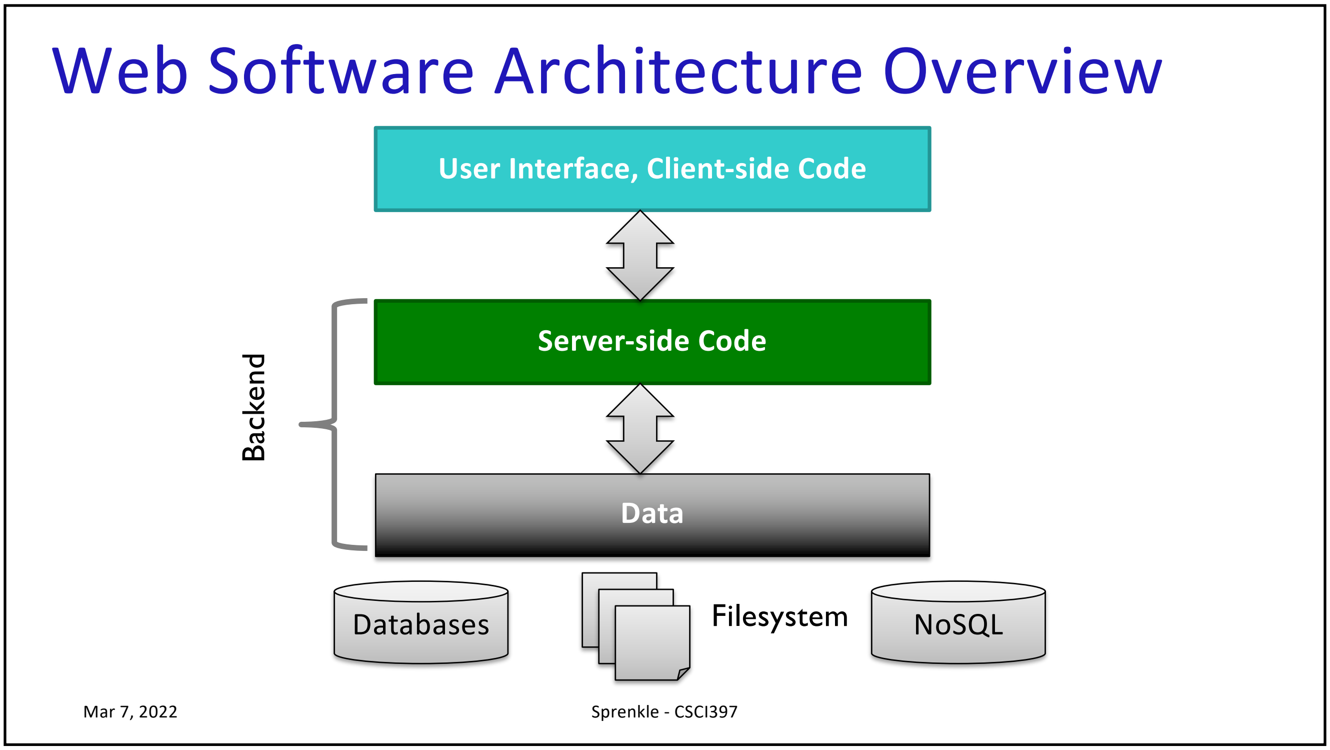
User Interface, Client-side Code

Server-side Code

Data

Backend

5



Web Software Architecture Overview

User Interface, Client-side Code

Server-side Code

Data

Backend

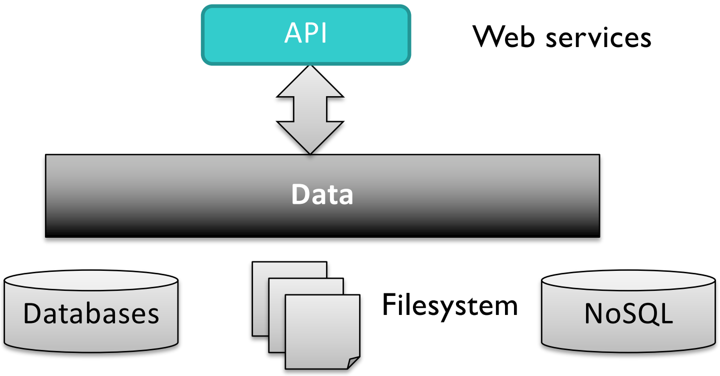
Databases

Filesystem

NoSQL

6

Web APIs



7

RELATIONAL DATABASES

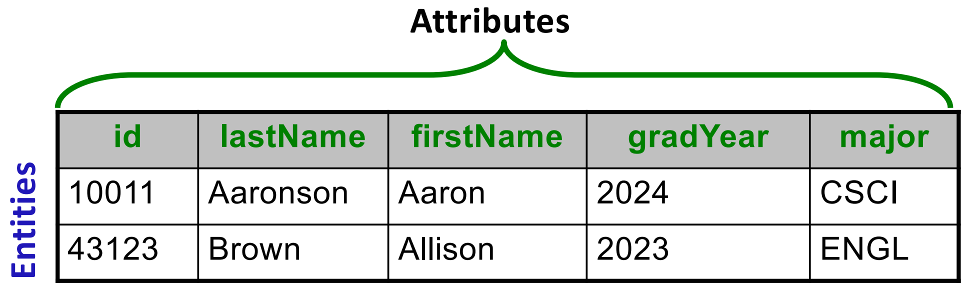
8

Database Overview

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

Example Students Table

- id is the *primary key*
- Attributes: Columns
- Entities: rows



Attributes

Entities

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2024	CSCI
43123	Brown	Allison	2023	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
  - 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 result as 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

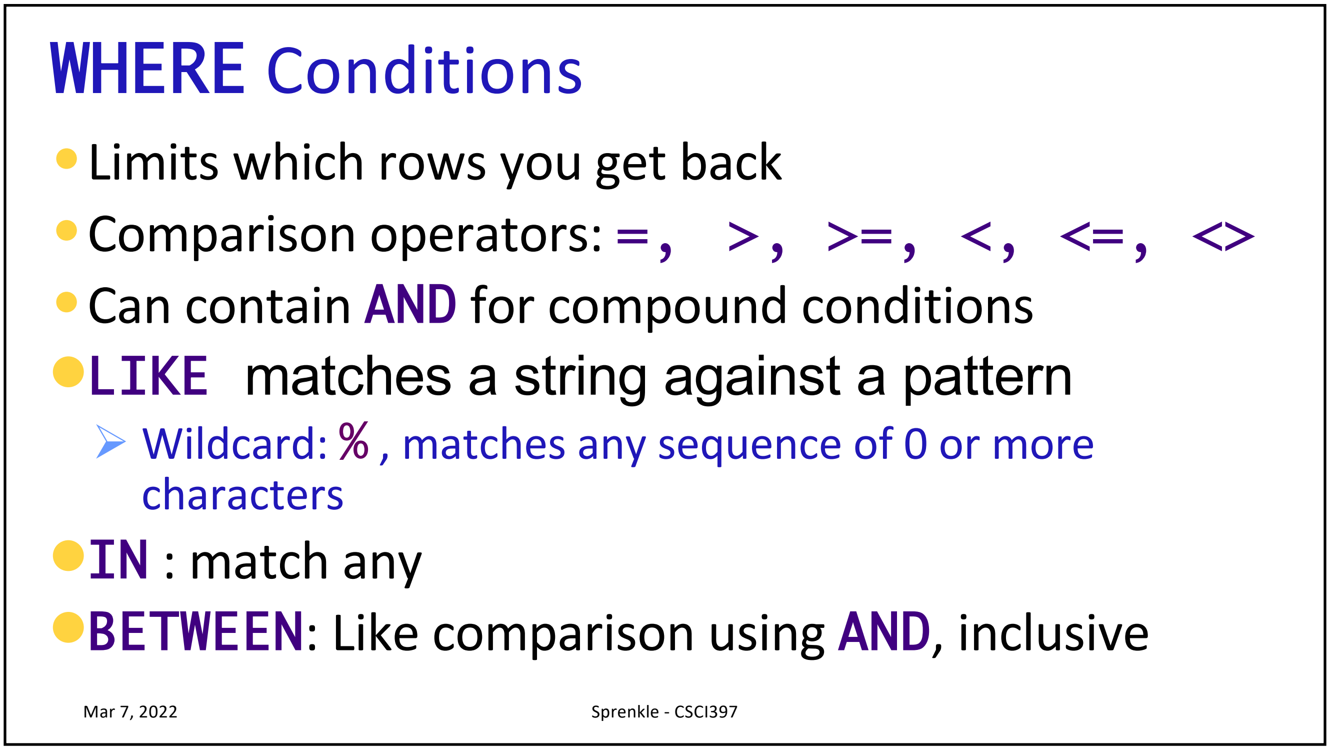
- `SELECT * FROM Students;`

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2018	CSCI
43123	Brown	Allison	2017	ENGL

- `SELECT lastName, major FROM Students;`

lastName	major
Aaronson	CSCI
Brown	ENGL

Virtual Tables

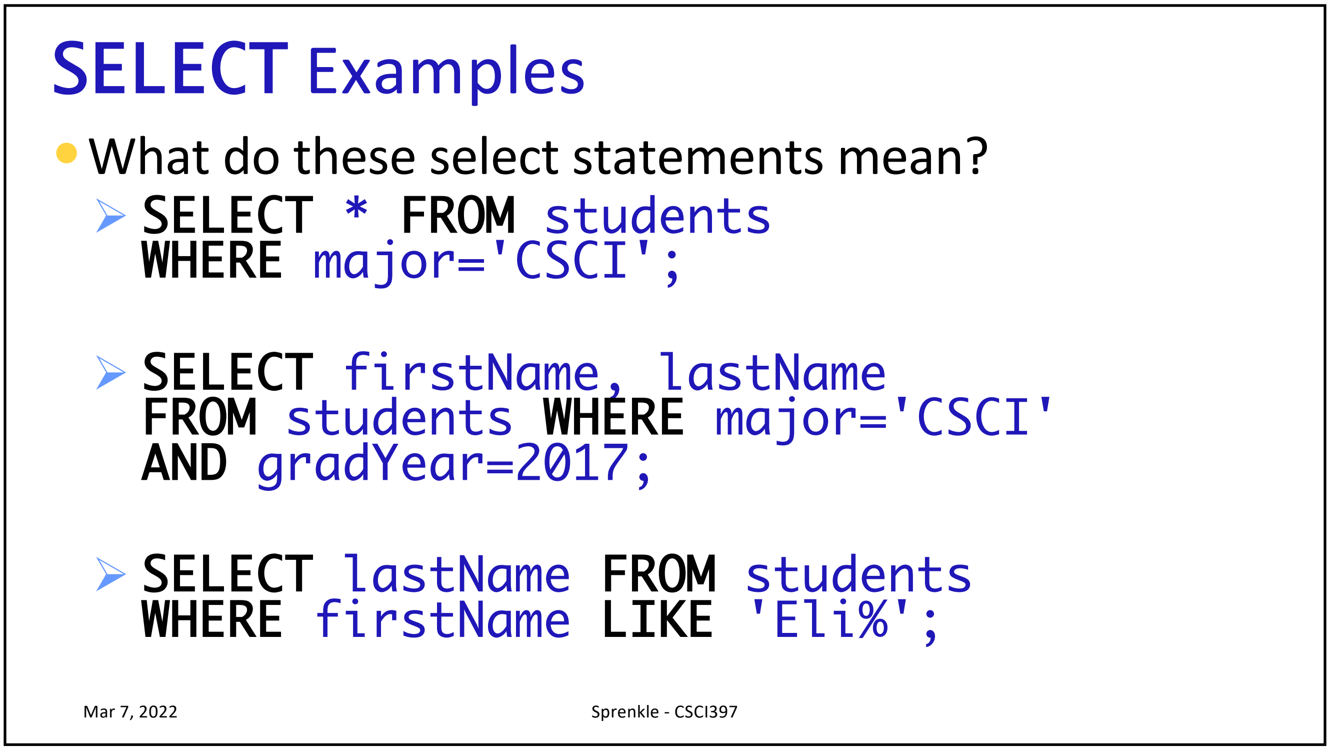


WHERE Conditions

- 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

Mar 7, 2022 Sprenkle - CSCI397

17



SELECT Examples

- What do these select statements mean?

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

  -
    ```
    SELECT firstName, lastName
    FROM students WHERE major='CSCI'
    AND gradYear=2017;
    ```

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

Mar 7, 2022 Sprenkle - CSCI397

18

SELECT Examples

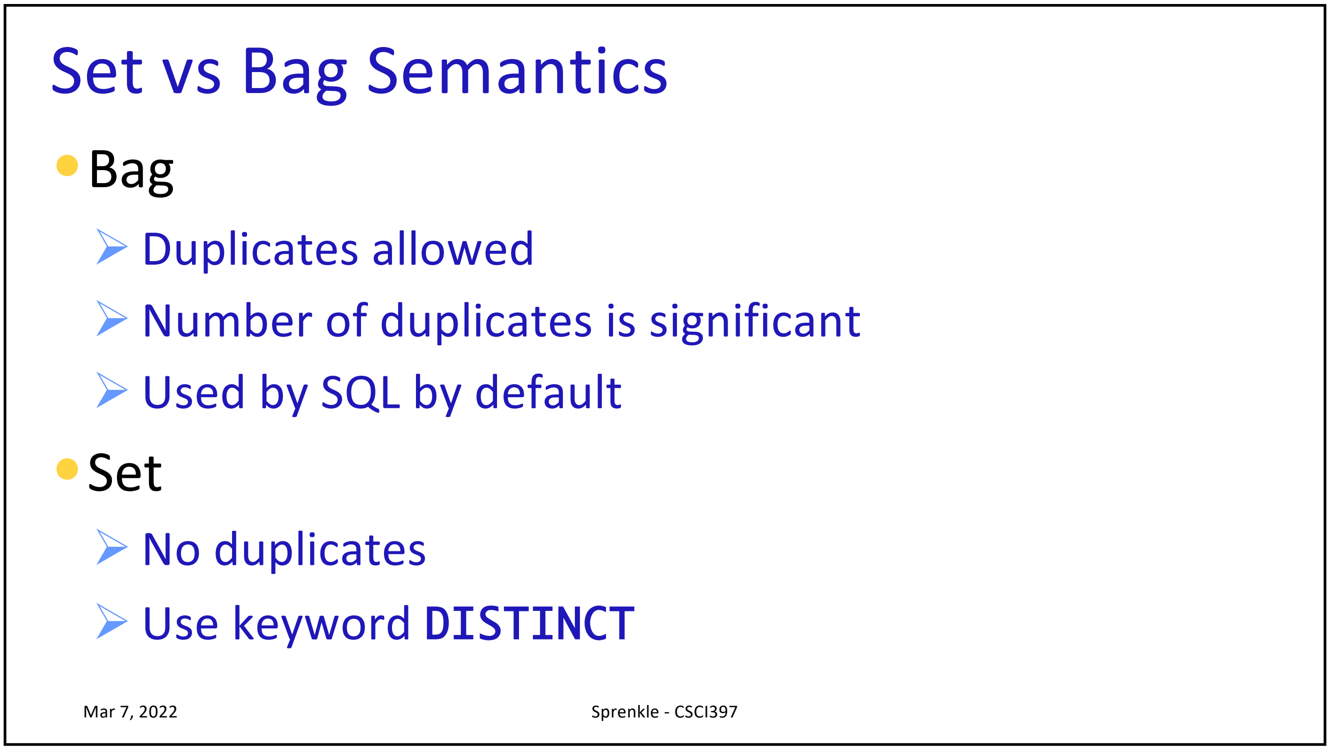
- What do these select statements mean?

  -
    ```
    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 2022 AND 2025;
    ```

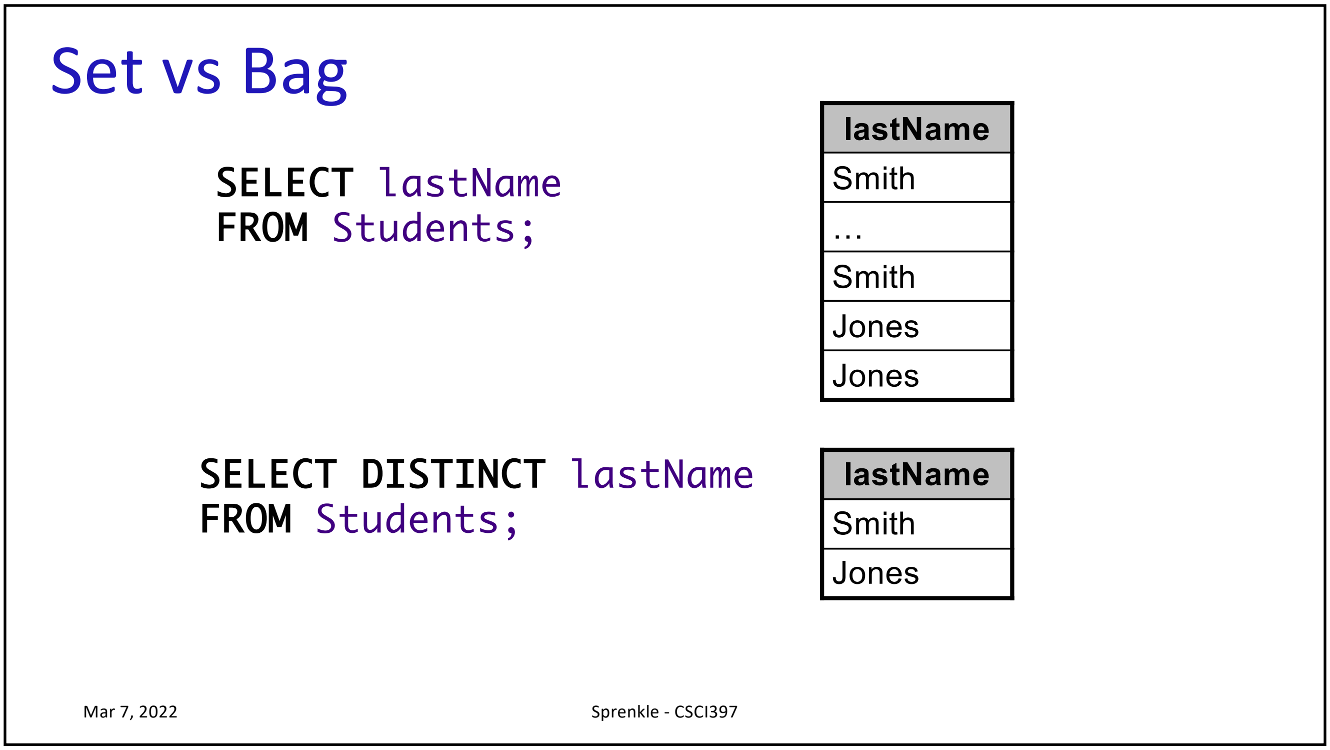
Set vs Bag Semantics



Set vs Bag Semantics

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

21



Set vs Bag

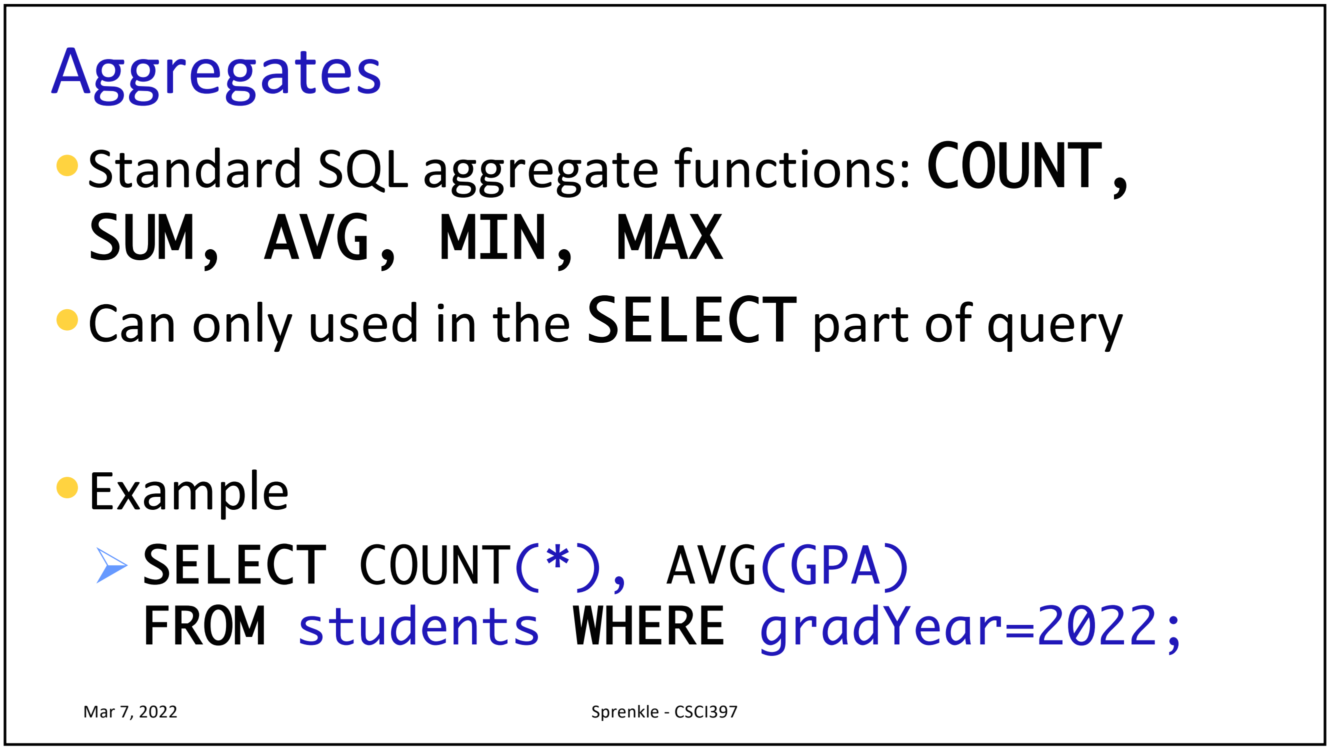
```
SELECT lastName
FROM Students;
```

lastName
Smith
…
Smith
Jones
Jones

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

lastName
Smith
Jones

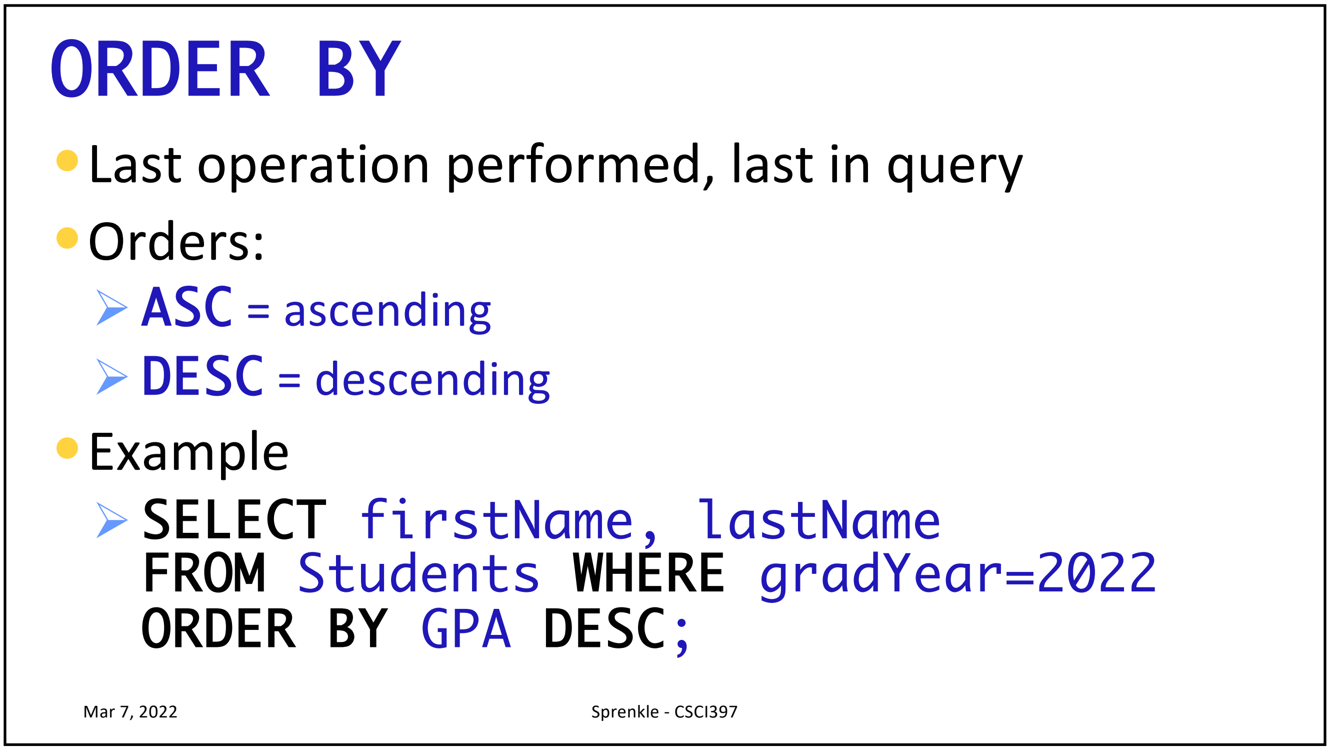
22



Aggregates

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

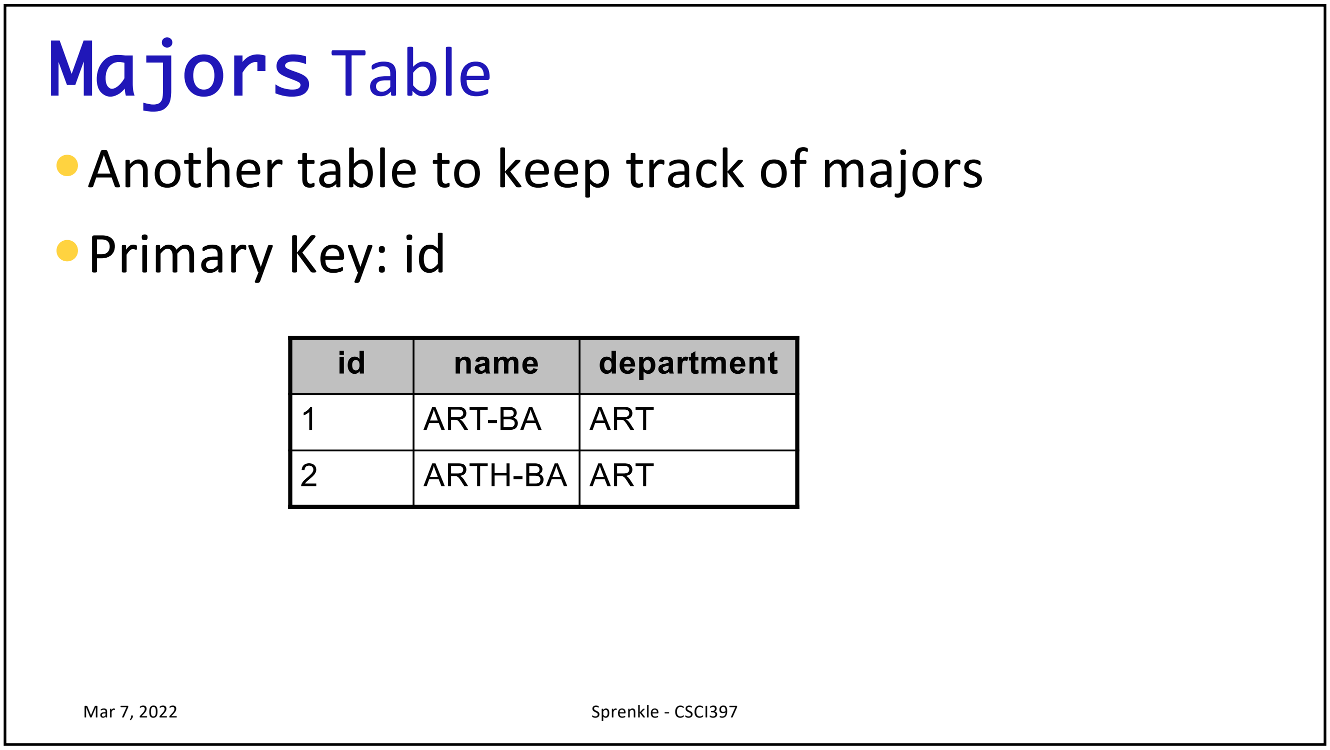
  -
    ```
    SELECT COUNT(*), AVG(GPA)
    FROM students WHERE gradYear=2022;
    ```



ORDER BY

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

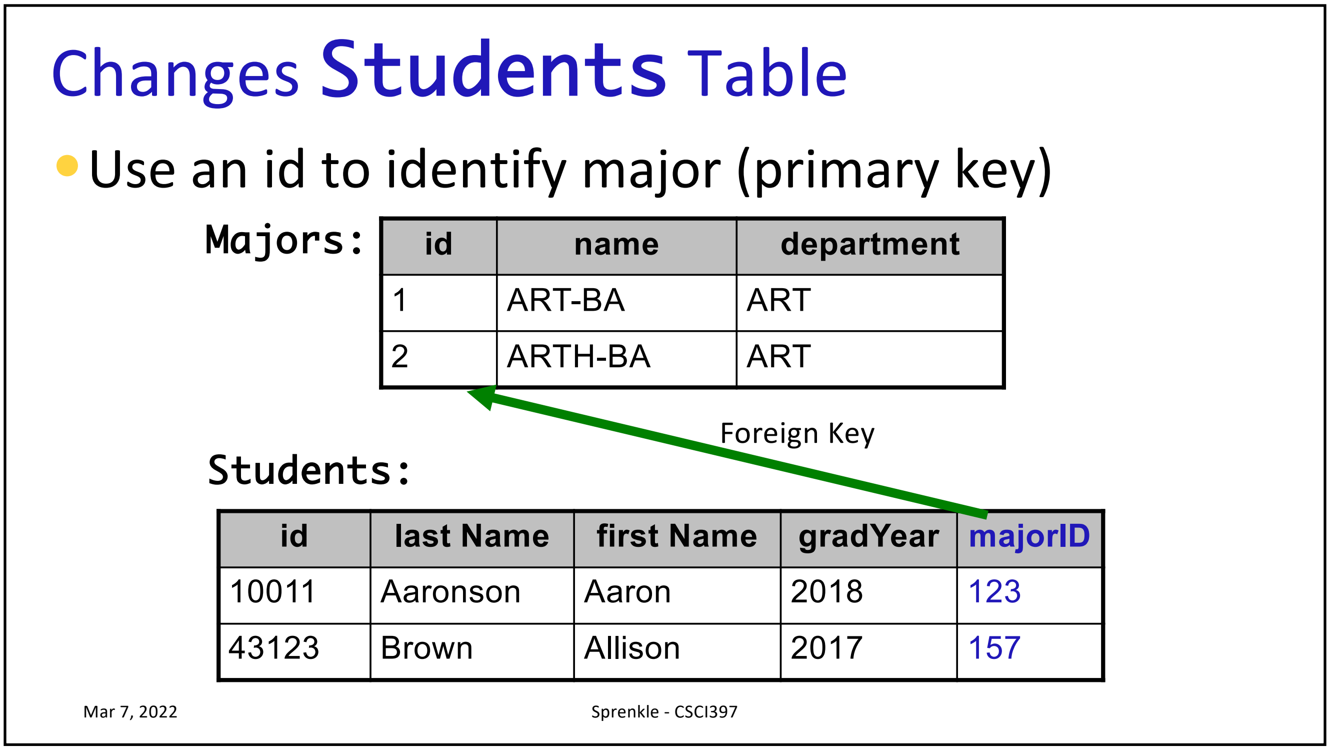
  -
    ```
    SELECT firstName, lastName
    FROM Students WHERE gradYear=2022
    ORDER BY GPA DESC;
    ```



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)

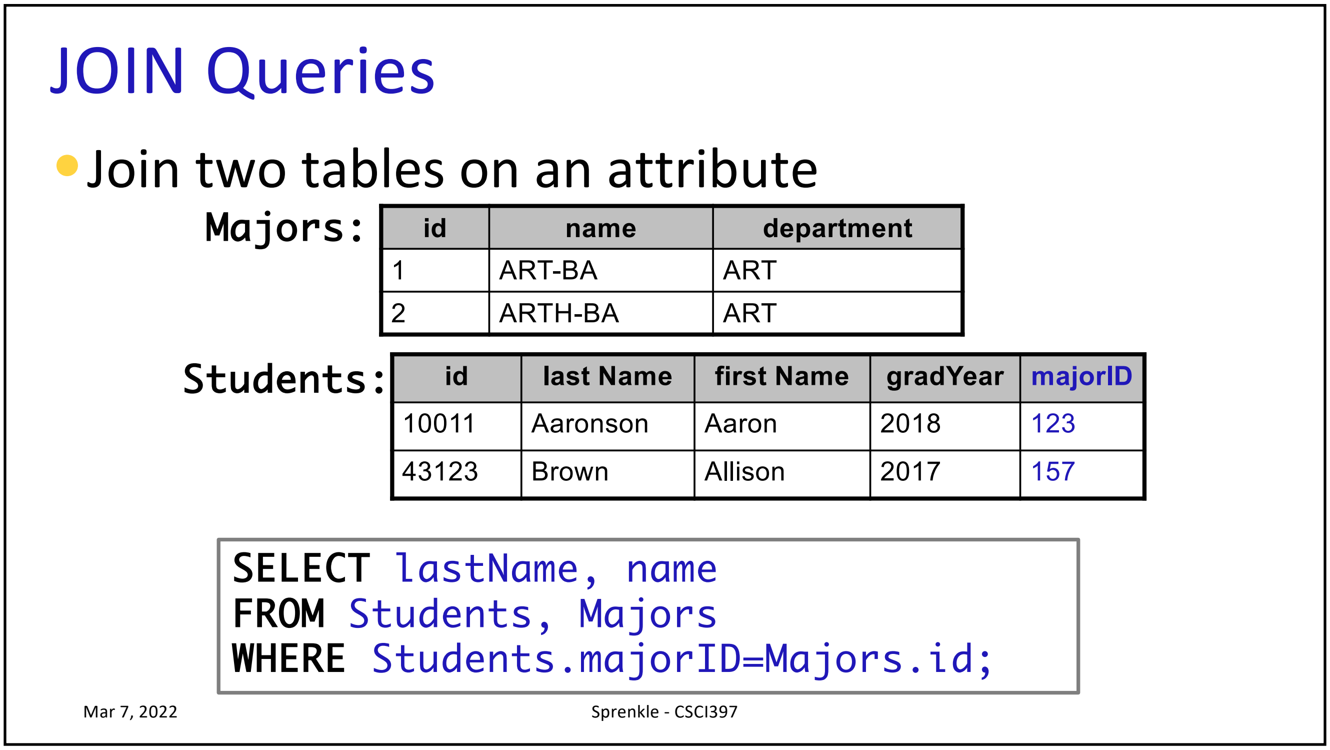
Majors:

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

Foreign Key

Students:

id	last Name	first Name	gradYear	majorID
10011	Aaronson	Aaron	2018	123
43123	Brown	Allison	2017	157



JOIN Queries

- Join two tables on an attribute

Majors:

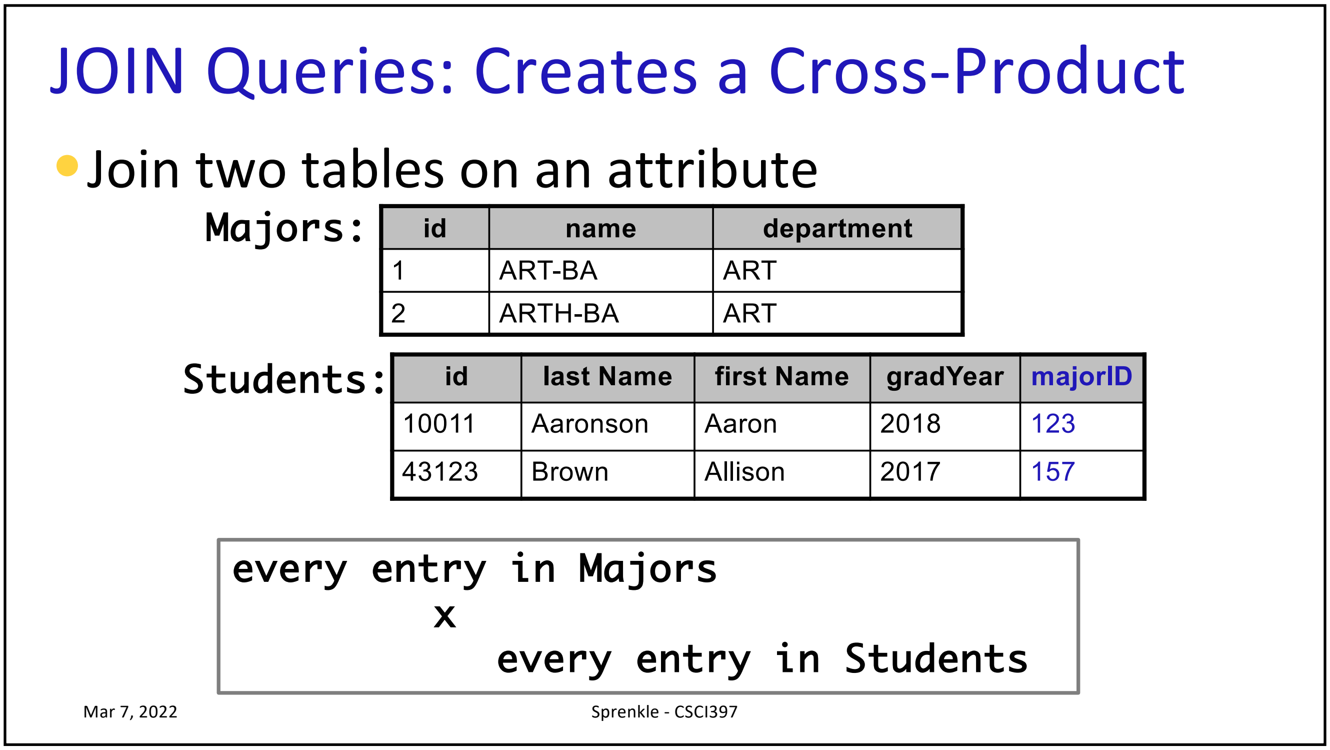
id	name	department
1	ART-BA	ART
2	ARTH-BA	ART

Students:

id	last Name	first Name	gradYear	majorID
10011	Aaronson	Aaron	2018	123
43123	Brown	Allison	2017	157

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

27



JOIN Queries: Creates a Cross-Product

- 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	2018	123
43123	Brown	Allison	2017	157

```
every entry in Majors
        x
           every entry in Students
```

28

Join Queries

Does a cross product of the joined tables

- Example:
  - Performing a select on 3 tables, each with two rows
  - `SELECT * FROM A, B, C`

A	B	C
A1	B1	C1
A2	B2	C2

- Results in this virtual table:

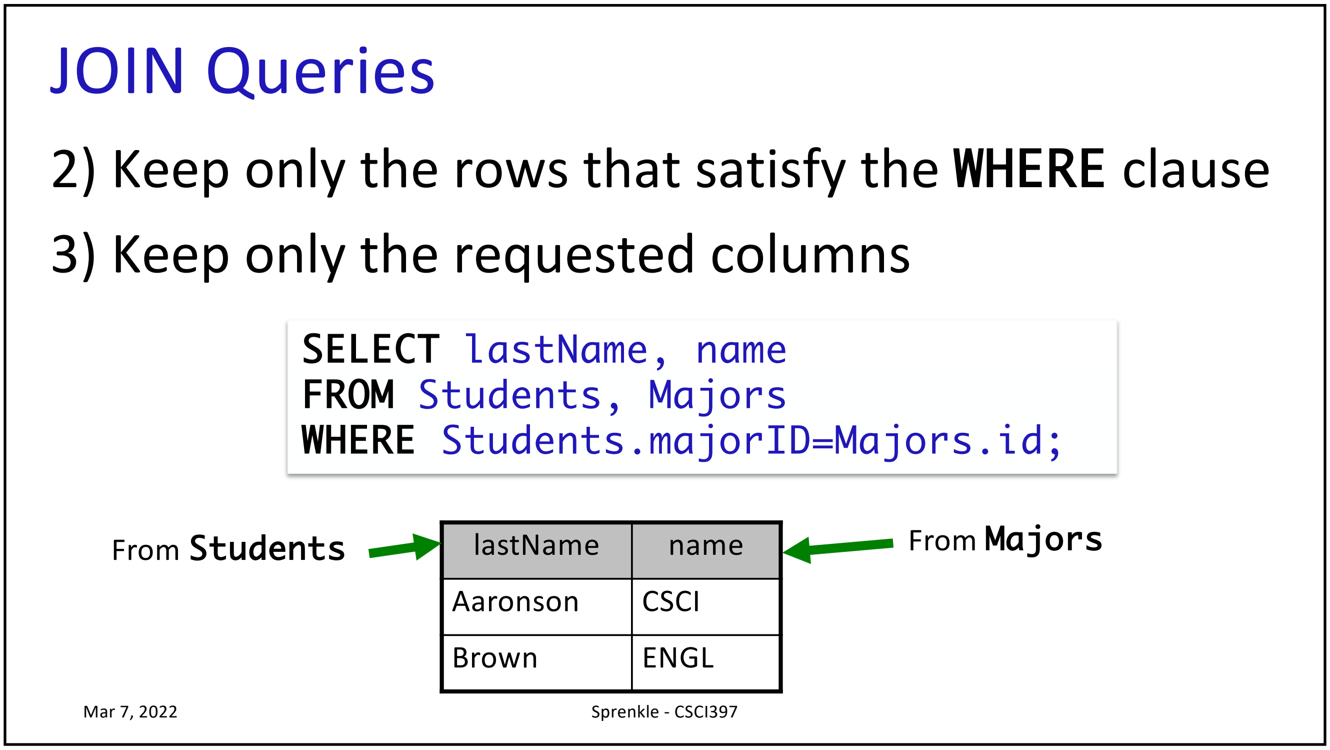
A1	B1	C1
A1	B1	C2
A1	B2	C1
A1	B2	C2
A2	B1	C1
A2	B1	C2
A2	B2	C1
…	…	…

Join Queries

1) Does a cross product of the joined tables

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

Id	Name	Dept	Id	LName	FName	…
M1			S1			
M1			S2			
M1			…			
M1			Sn			
M2			S1			
M2			S2			
M2			…			
M2			Sn			
…			…			



JOIN Queries

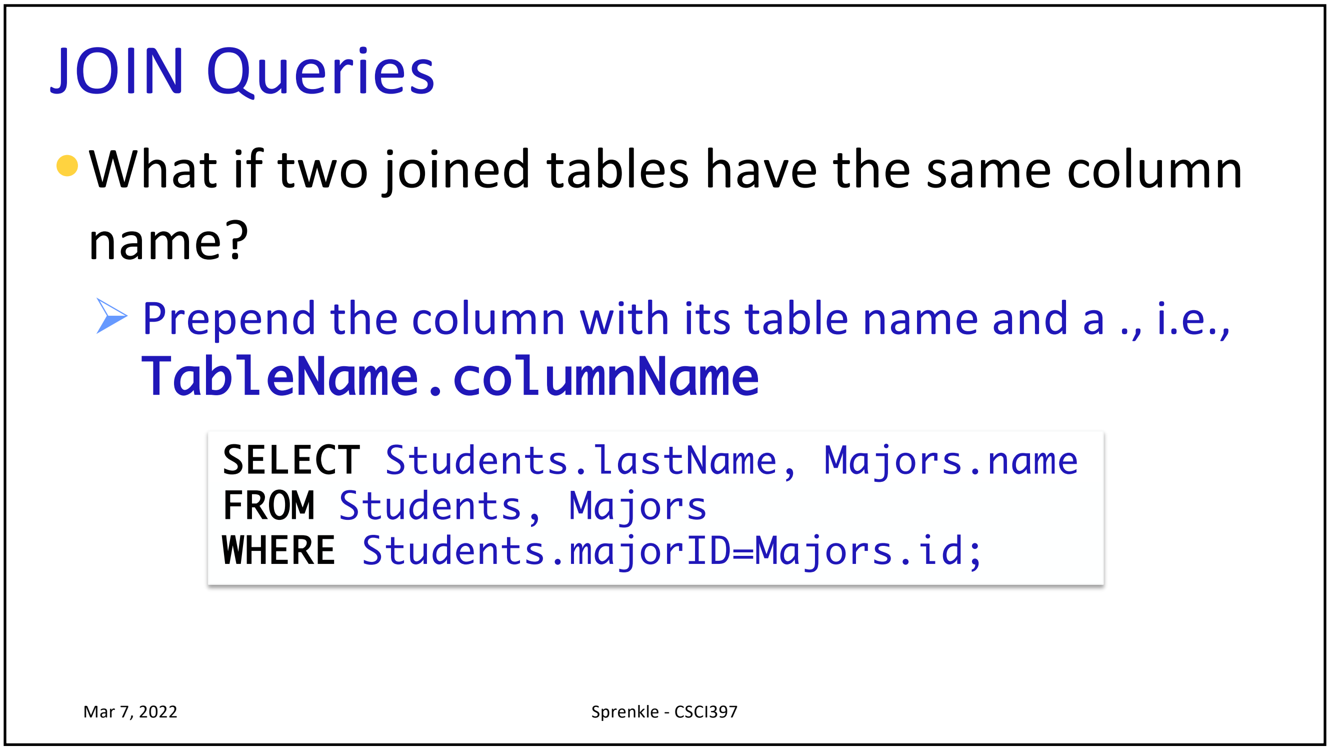
2) Keep only the rows that satisfy the **WHERE** clause

3) Keep only the requested columns

```
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 joined tables have the same column name?
  - Prepend the column with its table name and a ., 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

33

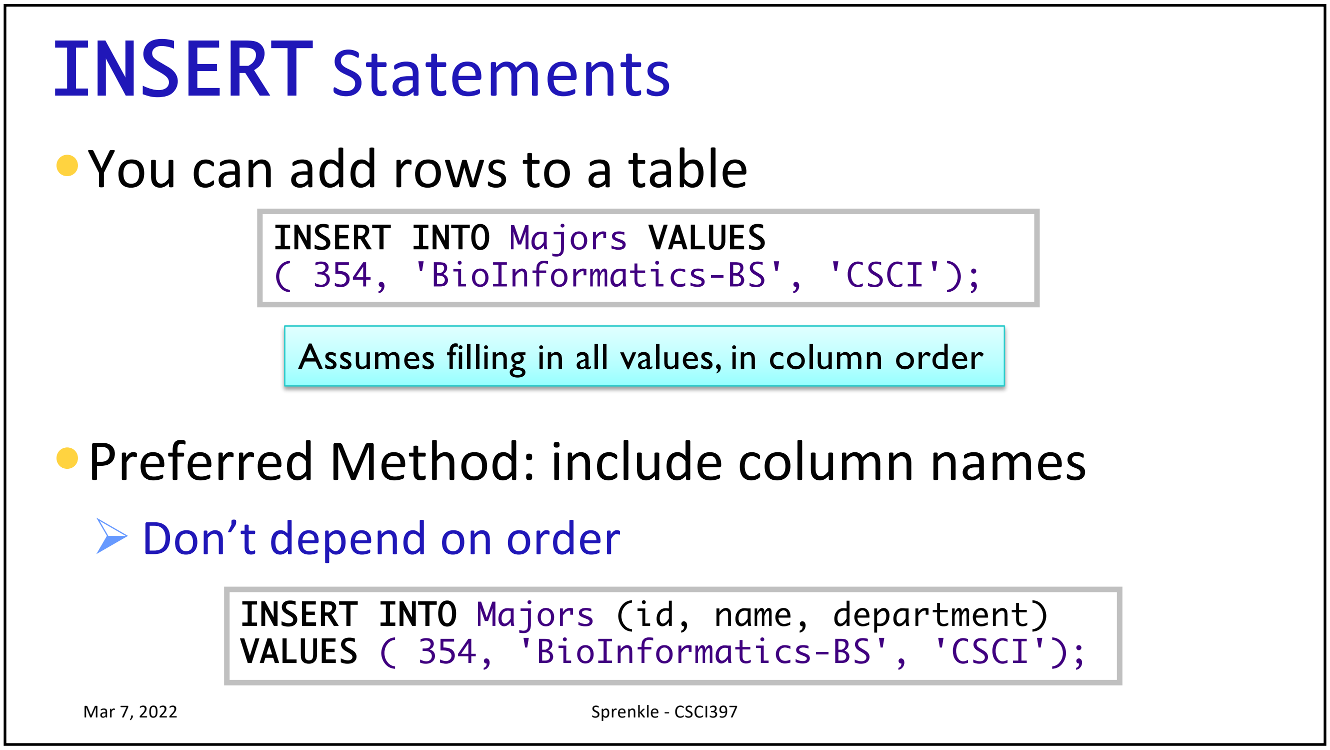
JOIN Query Example

- To find the students' majors with this new StudentsToMajors table, we would query

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

- Would create cross product of all 3 tables, then keep only the rows that satisfy the where clause, and only include the specified columns

34



INSERT Statements

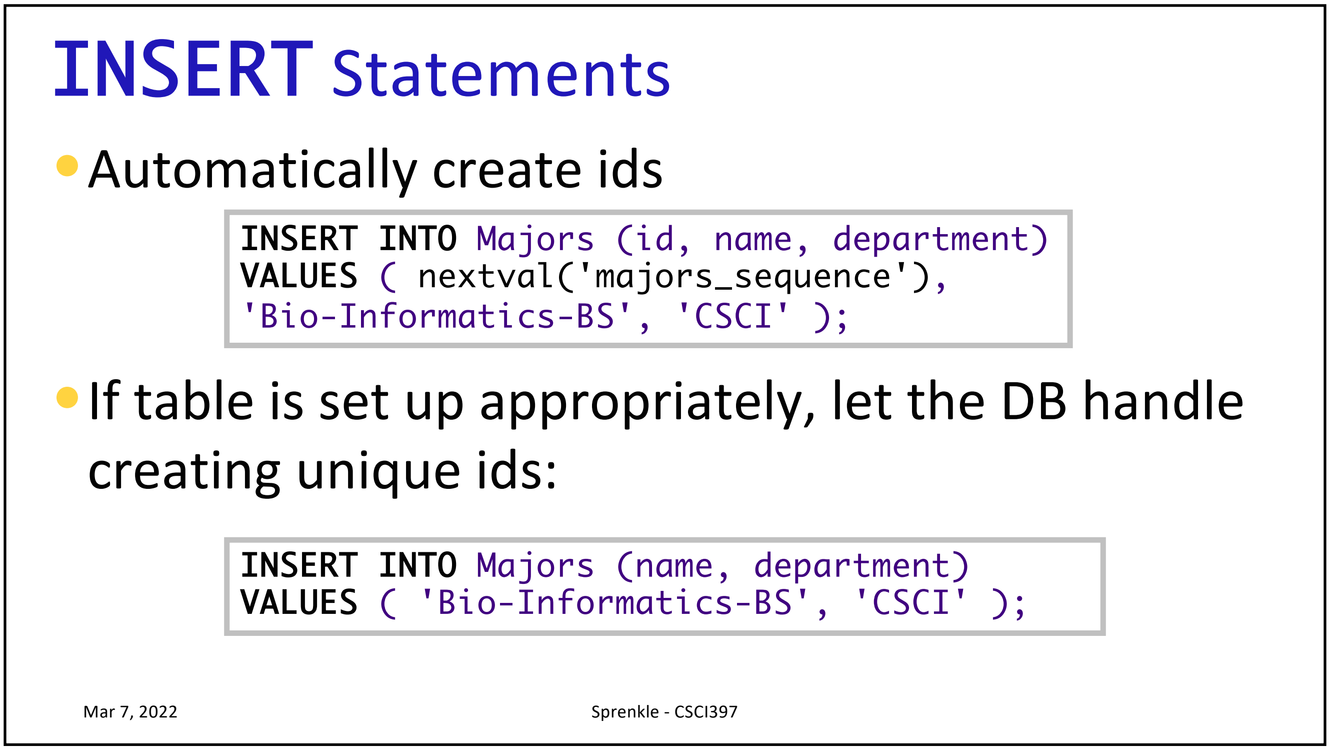
- You can add rows to a table

```
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=2025;
```

DELETE Statement

- You can delete rows from a table

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

- Example

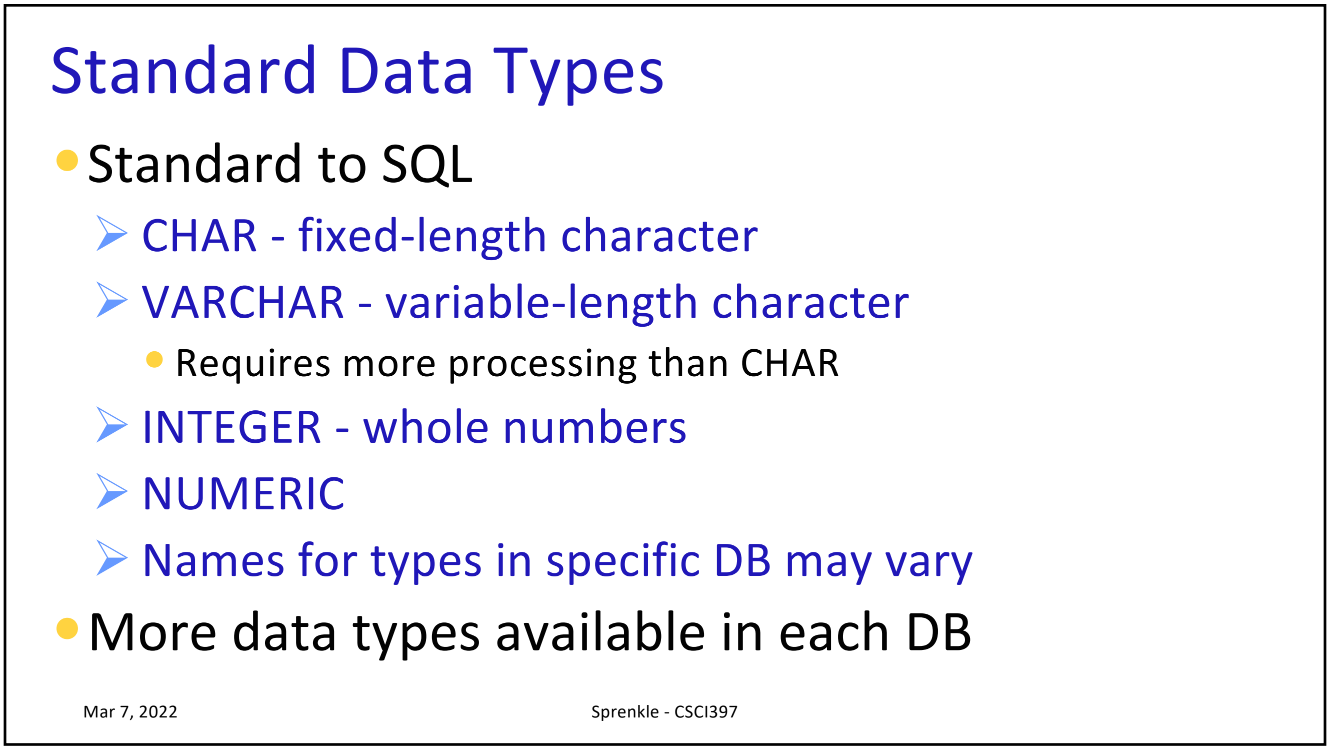
```
DELETE FROM EnrolledStudents WHERE
hasPrerequisites=False AND course_id=456;
```

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

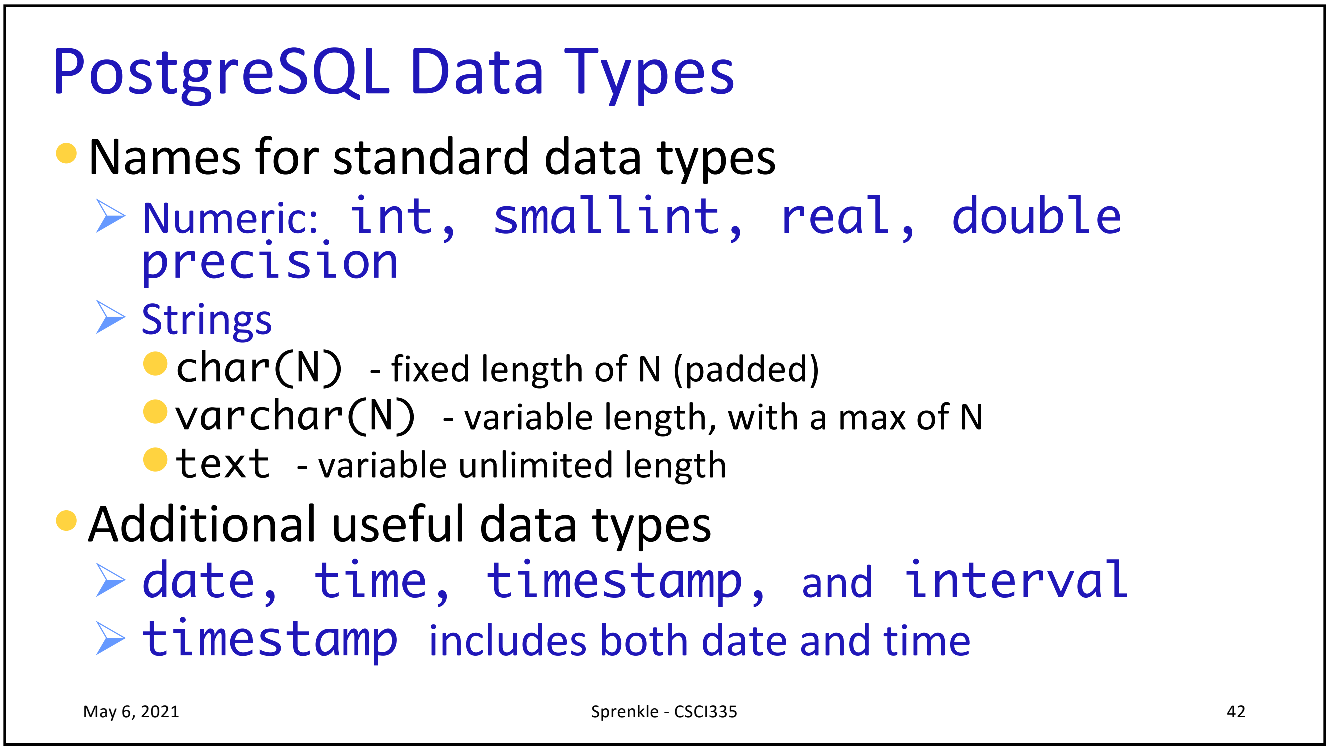


Standard Data Types

- 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

Mar 7, 2022 Sprenkle - CSCI397

41



PostgreSQL Data Types

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

May 6, 2021 Sprenkle - CSCI335 42

42

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
);
```

Interfacing with a Database

- Interactive mode
  - Run client
  - Enter SQL statements, one at a time

```
psql dbname
```

- Batch mode/command-line
  - Script/file of SQL commands
  - Direct to database

```
psql dbname < mycmds.sql
```

- Programming Language APIs
  - Examples: JDBC, psycopg2

Looking Ahead

- Anthony Danalis on PAPI
  - See Canvas for assignment, questions
- Friday: Data Center tour
  - Meet up there (behind law school parking lot) at the time you picked
- Using Docker to try out database