# Query Fundamentals 1

CSC 256, SQL Programming

# Example Relation

- We will use the following `"student"` table as a running example

| `first_name` | `last_name` | `location` | `gpa` |
|---|---|---|---|
| Spike | Spiegel | Mars | 2.0 |
| Jet | Black | Ganymede | 3.0 |
| Faye | Valentine | Earth | 2.5 |
| Edward | Wong | Earth | 4.0 |

# SELECT FROM

- `SELECT` retrieves information from a database; this is a read-only operation
- Basic Syntax:

```
SELECT column_name1, column_name2, ...
FROM table_name;
```

- Syntax to select all columns from a table:

```
SELECT * FROM table_name;
```

# SELECT Example

```
select * from student;

 first_name | last_name | location | gpa
------------+-----------+----------+-----
 Spike      | Spiegel   | Mars     |   2
 Jet        | Black     | Ganymede |   3
 Faye       | Valentine | Earth    | 2.5
 Edward     | Wong      | Earth    |   4
(4 rows)
```

- Note: SQL keywords are case insensitive

# SELECT Example

```
select first_name, last_name
from student;

 first_name | last_name
------------+-----------
 Spike      | Spiegel
 Jet        | Black
 Faye       | Valentine
 Edward     | Wong
(4 rows)
```

# Aliasing (AS)

- The AS keyword can rename a column in the result.
- Basic syntax

```
SELECT column_name AS new_name
FROM table_name;
```

- Note: if the new column name has one or more spaces, the it must be surrounded by double quotes.
- In general, double quotes for column/table names, single quotes for strings

# Derived Columns

- A derived (or computed column) column that is the result of an expression
- Example

```
select first_name, last_name, gpa + 10
from student;

 first_name | last_name | ?column?
------------+-----------+----------
 Spike      | Spiegel   |       12
 Jet        | Black     |       13
 Faye       | Valentine |     12.5
 Edward     | Wong      |       14
(4 rows)
```

- By default a derived column is given a arbitrary name; a specific name can be given with the AS keyword

# Derived Columns (continued)

- We can use arbitrary expressions and use parentheses to enforce order of operations
- Example:

```
select first_name, last_name, gpa + (10 * 0.25) as num
from student;

 first_name | last_name | num
------------+-----------+-----
 Spike      | Spiegel   | 4.5
 Jet        | Black     | 5.5
 Faye       | Valentine |   5
 Edward     | Wong      | 6.5
(4 rows)
```

# Expressions

- We don't necessarily need columns; we can use SQL as a calculator

```
select
  3 * 3 as "multiplication",
  6 + 6 as "addition",
  1 - 1 as "subtraction",
  10 / 2 as "division";
```

```
 multiplication | addition | subtraction | division
----------------+----------+-------------+----------
              9 |       12 |           0 |        5
(1 row)
```

# Text Concatenation

- The || operator is used to concatenate text
- Example:

```
select first_name || ' ' || last_name as "Full Name"
from student;

   Full Name
---------------
 Spike Spiegel
 Jet Black
 Faye Valentine
 Edward Wong
(4 rows)
```

# Filtering and WHERE

- The `WHERE` clause filters rows that satisfy some criteria
- Basic Syntax:

```
SELECT column_name1, column_name2, ...
FROM table_name
WHERE column_name1 operator value;
```

# WHERE Example

```
select first_name, last_name, location
from student
where location = 'Mars';

 first_name | last_name | location
------------+-----------+----------
 Spike      | Spiegel   | Mars
(1 row)
```

- Note: text comparisons are case sensitive

# Basic `WHERE` Clause Operators

- `=`: equal
- `<>`: not equal
- `>`: greater than
- `<`: less than
- `>=`: greater than or equal to
- `<=`: less than or equal to
- `AND`: logical and
- `OR`: logical or
- `BETWEEN`: between an inclusive range
- `LIKE`: search for a pattern
- `IN`: specify multiple possible values for a column

# AND, OR, and NOT

- The `AND` operator retrieves a record if both the first condition *and* the second condition are true
- The `OR` operator retrieves a record if either the first condition *or* the second condition are true
- The `NOT` operator retrieves a record if the condition is false
- Note that precedence can be confusing; we should use parentheses to be clear

# AND Example

```
select *
from student
where (gpa > 2.0) and (gpa < 4.0);

 first_name | last_name | location | gpa
------------+-----------+----------+-----
 Jet        | Black     | Ganymede |   3
 Faye       | Valentine | Earth    | 2.5
(2 rows)
```

# OR Example

```
select *
from student
where (gpa = 2.0) or (gpa = 4.0);

 first_name | last_name | location | gpa
------------+-----------+----------+-----
 Spike      | Spiegel   | Mars     |   2
 Edward     | Wong      | Earth    |   4
(2 rows)
```

# NOT Example

```
select *
from student
where not (location = 'Earth');

 first_name | last_name | location | gpa
------------+-----------+----------+-----
 Spike      | Spiegel   | Mars     |   2
 Jet        | Black     | Ganymede |   3
(2 rows)
```

# NULL handling

- `NULL` represents the absence of data
- This could mean:
  - There is no value, or
  - There is a value but it is unknown
- We cannot check for a NULL value with the equality (=) operator, we must use `IS NULL` (or `IS NOT NULL`).

# NULL Example

- First, let us add a record with a `NULL` value

| first_name | last_name | location | gpa |
|---|---|---|---|
| Spike | Spiegel | Mars | 2.0 |
| Jet | Black | Ganymede | 3.0 |
| Faye | Valentine | Earth | 2.5 |
| Edward | Wong | Earth | 4.0 |
| Ein | NULL | Mars | 3.9 |

# NULL Example

```
select *
from student
where last_name is null;

 first_name | last_name | location | gpa
------------+-----------+----------+-----
 Ein        |           | Mars     | 3.9
```

# NULL and Logical Operators

With respect to logical operators and `NULL`, we get a three-valued logic

| p | q | p `AND` q | p `OR` q | `NOT` p |
|---|---|---|---|---|
| true | true | true | true | false |
| true | false | false | true | false |
| true | unknown | unknown | true | false |
| false | true | false | true | true |
| false | false | false | false | true |
| false | unknown | false | unknown | true |
| unknown | true | unknown | true | unknown |
| unknown | false | false | unknown | unknown |
| unknown | unknown | unknown | unknown | unknown |

# BETWEEN

- The `BETWEEN` operator selects values within a range
- Syntax:

```
SELECT column_name, column_name, ...
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

# The LIKE Operator

- The LIKE operator is used to search a column for a pattern
- Syntax:

```
SELECT column_name1, ...
FROM table_name
WHERE column_name1 LIKE pattern;
```

- Wilcard characters:
  - % matches zero or more characters
  - _ match 1 character

# The IN Operator

- The IN operator specifies multiple values in a WHERE clause
- Syntax:

```
SELECT column_name1, ...
FROM table_name
WHERE column_name IN (value1, value2, ...);
```