Query Fundamentals 2

CSC 256, SQL Programming

Example Relation (from previous lecture)

- 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
Ein	NULL	Mars	3.9

ORDER BY

- `ORDER BY` sorts the records in ascending or descending order; note that there is no guaranteed sort order in SQL
- Syntax:

```
SELECT column_name1, column_name2, ...
FROM table_name
ORDER BY column1 [ASC|DESC] , column2 [ASC|DESC], ... ;
```

- Note that in `ASC|DESC` the vertical bar (`|`) means or, so you can choose ascending or descending order, but not both.
- The square brackets (`[ ]`) means optional; default sorting order is ascending.

ORDER BY Example Default

```
select *
from student
order by first_name;

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

ORDER BY Example Explicit ASC

```
select *
from student
order by first_name asc;

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

ORDER BY Example Explicit DESC

```
select *
from student
order by first_name desc;

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

ORDER BY Example Multiple Columns

```
select *
from student
order by location asc, first_name desc;

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

ORDER BY (continued)

- We can use expressions in ORDER BY
- We can order by aliases
- Example:

```
select first_name, last_name, ceil(gpa) as gpa_rounded_
from student
order by gpa_rounded_up;

first_name  last_name  gpa_rounded_up
----------  ---------  --------------
Spike       Spiegel    2.0
Jet         Black      3.0
Faye        Valentine  3.0
Edward      Wong       4.0
Ein                    4.0
```

Aliases and WHERE

- Let us try the previous example and refer to an alias in the WHERE clause:

```
select first_name, last_name, ceil(gpa) as gpa_rounded_
from student
where gpa_rounded_up = 4
order by gpa_rounded_up;

first_name  last_name  gpa_rounded_up
----------  ---------  --------------
Edward      Wong       4.0
Ein         NULL       4.0
```

  - This works in SQLite, but does not in many other RDBMSs.

SELECT Order of Writing

- The clauses of a SELECT statement are written in the order:
  1. SELECT: select the columns to appear in the output
  2. FROM: pick tables to be queried
  3. WHERE: filter the rows
  4. GROUP BY: aggregate rows (next lecture)
  5. HAVING: filter the aggregates (next lecture)
  6. ORDER BY: sort the rows
  7. LIMIT: limit the number of rows returned (later in this lecture)
- Note: there are a few clauses not listed here that we will cover in the future

SELECT Order of Execution

- SELECT order of execution is different from how written lexically:
  1. FROM: pick tables to be queried
  2. WHERE: filter the rows
  3. GROUP BY: aggregate rows (next lecture)
  4. HAVING: filter the aggregates (next lecture)
  5. SELECT: select the columns to appear in the output
  6. ORDER BY: sort the rows
  7. LIMIT: limit the number of rows returned (later in this lecture)
- NOTE: memorize this order; it will help you

Aliases and WHERE Fixed

- Simple fix: repeat the expression. (we will see a way to eliminate the duplication in a future lecture)

  ```
  select first_name, last_name, ceil(gpa) as gpa_rounded_
  from student where ceil(gpa) = 4
  order by gpa_rounded_up;

  first_name  last_name  gpa_rounded_up
  ----------  ---------  --------------
  Edward      Wong       4.0
  Ein         NULL       4.0
  ```

  - This will work with all popular RDBMSs.

LIMIT

- The LIMIT keyword limits the number of rows returned; if the number of rows returned is less than the LIMIT specified, then LIMIT does nothing
- Example:

```
select *
from student
limit 2;

first_name  last_name  location  gpa
----------  ---------  --------  ---
Spike       Spiegel    Mars      2.0
Jet         Black      Ganymede  3.0
```

- Note: because there is no guaranteed order for the result of a select, we should always use an ORDER BY with LIMIT.

OFFSET

- OFFSET skips a certain number of rows
- Example: (note this is PostreSQL output because this syntax is not supported in SQLite)

```
select *
from student
order by first_name
offset 2 rows fetch next 3 rows only;

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

FETCH

- The FETCH without an OFFSET acts like a LIMIT
- Example: (note this is PostreSQL output because this syntax is not supported in SQLite)

```
select *
from student
order by first_name
fetch next 3 rows only;

 first_name | last_name | location | gpa
------------+-----------+----------+-----
 Edward     | Wong      | Earth    |   4
 Ein        |           | Mars     | 3.9
 Faye       | Valentine | Earth    | 2.5
(3 rows)
```

LIMIT OFFSET (SQLite Extension)

- SQLite (and PostgreSQL) has a shorter syntax for LIMIT with an OFFSET
- Example:

```
select *
from student
order by first_name
limit 3 offset 2;

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

DISTINCT

- `DISTINCT` returns values with duplicates removed
- Syntax:

```
SELECT DISTINCT column_name, column_name, ...
FROM table_name;
```

- DISTINCT is executed as part of the select clause
- DISTINCT is executed after any expressions

DISTINCT Example

```
select distinct location
from student;

location
--------
Mars
Ganymede
Earth
```

DISTINCT Example (with expression)

```
select distinct ceil(gpa)
from student;

ceil(gpa)
---------
2.0
3.0
4.0
```

CASE Expressions

- A CASE expression is inline conditional logic (similar to ternary operator)
- CASE has two forms: simple and searched
- Simple syntax:

```
CASE input_expr
    WHEN expr1 THEN result1
    WHEN expr2 THEN result2
    ...
    [ELSE resultn]
END
```

- The **equality** conditions are checked in the order of definition. The first condition that evaluates to true is chosen. In the case of no match (when the optional else is not specified), NULL is returned.

CASE Example (simple syntax)

```
select
  first_name,
  case ceil(gpa)
    when 4 then 'great'
    when 3 then 'good'
    else 'poor' end as score
from student;
```

```
first_name  score
----------  -----
Spike       poor
Jet         good
Faye        good
Edward      great
Ein         great
```

CASE Expressions (continued)

- CASE searched form syntax:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    [ELSE resultn]
END
```

- The **arbitrary** conditions are checked in the order of definition. The first condition that evaluates to true is chosen. In the case of no match (when the optional else is not specified), NULL is returned.

CASE Example (searched form syntax)

```
select
  first_name,
  case
    when gpa >= 4 then 'great'
    when gpa between 2 and 3
    then 'good'
    else 'poor' end as score
from student;

first_name  score
----------  -----
Spike       good
Jet         good
Faye        good
Edward      great
Ein         poor
```

CASE Expressions (continued)

- CASE is an expression, so it can be used anywhere an expression can be used:
  - SELECT
  - ORDER BY
  - WHERE
  - and a few other places
- Note: we need to be careful with null values. The simple case form implicitly does an equality comparison, so we can not use that form when handling null values.

NULL Values and ORDER BY Example

- NULL values last

```
select *
from student
order by
  case when last_name is null then 1 else 0 end,
  last_name;

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

NULL Values and ORDER BY Example

- NULL values first

```
select *
from student
order by
  case when last_name is null then 0 else 1 end,
  last_name;
```

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

NULL Values and ORDER BY (SQLite Extension)

- SQLite (and PostgreSQL) has an alternative syntax for sorting null values which is specified as `NULLS FIRST` or `NULLS LAST`
- Example:

```
select *
from student
order by last_name nulls last;

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