CSE462/562: Database Systems (Spring 22)

Lecture 4: SQL

2/10/2022

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Structured Query Language (SQL)

- SQL stands for Structured Query Language
  - It's not only a "query language"
  - Consists of
    - Data Definition Language (DDL): define/modify schema, delete relations
      - Integrity checks: foreign-key constraints, general constraints, triggers
      - View definition, authorization specification, …
    - Data Manipulation Language (DML): query/insert/update/delete in a DB instance
    - Transaction control
    - Stored procedure, embedded SQL, SQL Procedural language, …

- The most widely used relational query language. Latest standard is SQL-2016
  - Each DBMS (e.g. MySQL/PostgreSQL) has some "unique" aspects
  - We'll only review the basics of SQL.

DDL - Create Table

- `CREATE TABLE table_name ( {`
  `column_name data_type`
  `} [,…])`

- Data Types include:
  - `CHAR(n)` – fixed-length character string
  - `VARCHAR(n)` – variable-length character string with max length n
  - `SMALLINT, INTEGER, BIGINT` – signed 2/4/8-byte integers
  - `NUMERIC[(p[,s])]` – exact numeric of selectable precision
  - `REAL, DOUBLE` – single/double floating point numbers
  - `DATE, TIME, TIMESTAMP, …`
  - `SERIAL` - unique ID for indexing and cross reference
  - …

DDL - Create Table w/ Column Constraints

- CREATE TABLE *table_name* ({
  column_name data_type
  [*column_constraint* [, ...]]
  } [,…])

- Column Constraints:

```
[CONSTRAINT constraint_name]  {
    DEFAULT default_expr |
    NOT NULL | NULL | UNIQUE | PRIMARY KEY |
    CHECK (boolean_expression) |
    REFERENCES reftable [(refcolumn)] [ON DELETE action]
      [ON UPDATE action ] }
```

where *action* is one of:

NO ACTION, CASCADE, SET NULL, SET DEFAULT

can only reference the column's value

DDL - Create Table w/ Table Constraints

- CREATE TABLE *table_name* ({
  column_name data_type
  [*column_constraint* [, ...]] |
  table_constraint
  } [,…])

- Table constraints:

[CONSTRAINT *constraint_name*]{
  UNIQUE (*column_name* [, ...]) |
  PRIMARY KEY (*column_name* [, ...]) |
  CHECK (*boolean_expression*) |
  FOREIGN KEY (*column_name* [, ...])
    REFERENCES *reftable* [(*refcolumn* [, ...])]
    [ON DELETE *action*] [ON UPDATE *action*]}

where *action* is one of:

NO ACTION, CASCADE, SET NULL, SET DEFAULT

can only reference multiple table column's values

DDL -Create Table (Examples)

- ```
  CREATE TABLE student (
       sid        INTEGER PRIMARY KEY,
       name       VARCHAR(100) NOT NULL,
       login      VARCHAR(32) UNIQUE NOT NULL,
       major      VARCHAR(3),
       adm_year DATE);
  ```

- ```
  CREATE TABLE enrollment(
       sid        INTEGER REFERENCES student ON DELETE SET NULL
       semester VARCHAR(3),
       cno        INTEGER,
       grade      NUMERIC(2, 1)
       PRIMARY KEY (sid, semester, cno));
  ```

Other DDL statements

- `DROP TABLE table_name;`
- `ALTER TABLE table_name action [,…];`
  where *action* is one of
  `ADD column_name data_type [column_constraints [,…]]`
  `DROP column_name data_type`
  `ALTER coumn_name …`
  `ADD table_constraint`
  `DROP CONSTRAINT constraint_name`
  …

SQL DML

- `SELECT` statement
- `INSERT` statement
- `DELETE` statement
- `UPDATE` statement

SQL DML Semantics

- SQL uses multi-set relational algebra by default
  - Multi-set semantics (i.e., allow duplicate rows), let Q, Q' be multi-set RA queries
    - For projection $\pi_A Q$, no deduplication over the attribute set A
    - For selection $\sigma_P Q$, all copies of rows in Q that satisfies predicate P are retained
    - For cross product $Q \times Q'$, there are cc' copies of $t \circ t'$ if there are c copies of t in Q and c' copies of t' in Q'
    - Deduplications are explicit via `distinct` keyword
    - Set union, set difference and set intersection, see later discussion
  - SQL also supports operators that can't be expressed in the standard multi-set relational algebra
    - sorting
    - aggregation

Single-Table Query

- Single-table queries are straight-forward.

- To find all students admitted in 2021, we can write

```
SELECT *
FROM students S
WHERE S.adm_year = 2021;
```

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

result

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
102	Charlie	charlie7	CS	2021

Multi-Table Query

- We can express a join as follows

```
SELECT S.name, E.grade
FROM student S, enrollment E
WHERE S.sid=E.sid AND E.cno=562;
```

or

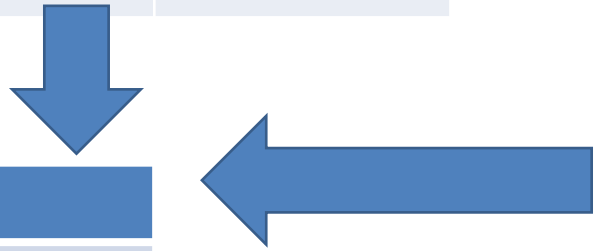
```
SELECT S.name, E.grade
FROM student S JOIN enrollment E
  ON S.sid = E.sid
WHERE E.cno = 562;
```

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0



Result

name	grade
Alice	2.0
Charlie	2.3

SQL Query Syntax

- `SELECT` and `FROM` clauses are mandatory
- `WHERE` clause is optional

```
SELECT  [DISTINCT] target-list
FROM    relation-list
[WHERE predicate]
```

- *`relation-list`*: a list of relation
  - each possibly with a table alias (aka correlation name)
- *`target-list`*: a list of expressions that may reference columns in the relation list
  - "`*`" to denote all the columns in the relation list
  - each may be renamed with `AS` clause (e.g., `S.name as student_name`)
  - `DISTINCT`: an optional keyword to deduplicate the result
- *`predicate`*: boolean expressions over the columns in the relation list, may contain
  - comparisons such as <, >, <=, >=, =, <>, LIKE
  - AND/OR/NOT
  - nested query
  - …

SQL supports string matching operator LIKE:
`_' stands for any one character and `%' stands for 0 or more arbitrary characters.
e.g., dname LIKE '%Engineering' will match all departments that ends with "Engineering" in its name

SQL Query Semantics

- A SQL query may be translated into the following multi-set relational algebra
  Let $R_1, R_2, \ldots, R_n$ be relations in the relation list
  and $E_1, E_2, \ldots, E_m$ be the expressions in the target list
  and P be the boolean predicate in the WHERE clause (P = true if WHERE clause is missing)

$$\pi_{E_1,E_2,\ldots E_m}\sigma_P R_1 \times R_2 \times \cdots \times R_n$$

- If there's `DISTINCT` keyword in the select clause
  - The final projection uses set semantics (in practice, implemented as a *deduplication* operator)
- This is a conceptual and probably the least efficient way of computing a SQL query
  - Query optimizer will find more efficient strategies that produce *the same result*

A running example

```
SELECT S.name, E.grade
FROM student S, enrollment E
WHERE S.sid=E.sid AND E.cno=562;
```

student S

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

enrollment E

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

$S \times E$

S.sid	name	login	major	adm_year	E.sid	semester	cno	grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
100	Alice	alicer34	CS	2021	102	s22	562	2.3
100	Alice	alicer34	CS	2021	100	f21	560	3.7
100	Alice	alicer34	CS	2021	100	s22	562	3.3
More results follows ……								

A running example (cont'd)

```
SELECT S.name, E.grade
FROM student S, enrollment E
WHERE S.sid=E.sid AND E.cno=562;
```

S.sid	name	login	major	adm_year	E.sid	semester	cno	grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
100	Alice	alicer34	CS	2021	102	s22	562	2.3
100	Alice	alicer34	CS	2021	100	f21	560	3.7
100	Alice	alicer34	CS	2021	100	s22	562	3.3
More results follows ……								

$\sigma_{S.sid=E.sid\ and\ E.cno=562} S \times E$

S.sid	name	login	major	adm_year	E.sid	semester	cno	grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
102	Charlie	charlie7	CS	2021	102	s22	562	2.3

A running example (cont'd)

```
SELECT S.name, E.grade
FROM student S, enrollment E
WHERE S.sid=E.sid AND E.cno=562;
```

S.sid	name	login	major	adm_year	E.sid	semester	cno	grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
102	Charlie	charlie7	CS	2021	102	s22	562	2.3

$$\pi_{S.name,E.grade}\sigma_{S.sid=E.sid\ and\ E.cno=562}S \times E$$

Final result =

name	grade
Alice	2.0
Charlie	2.3

ORDER BY Clause

- Optional ORDER BY clause sorts the final results before presenting them to the end user
  - `expr` is some expression of the columns in the relation list
  - Sort lexicographically
  - May also use positional notation (1, 2, 3, …)
    - denotes expr in target list
  - Default is ascending order `ASC`
    - Specify `DESC` for descending order

```
SELECT  [DISTINCT] target-list
FROM    relation-list
[WHERE predicate]
[ORDER BY] expr [ASC|DESC] [,…]
```

- Examples
  - `ORDER BY E.grade DESC` -- sort by descending order in grade
  - `ORDER BY 2 DESC` -- same as above
  - `ORDER BY E.grade DESC, S.name`
    - sort by descending grade first; then for equal values of grade, sort by name in ascending order
  - `ORDER BY 2 DESC, 1 ASC` -- same as above

Nested Query

- Nested queries may appear in `FROM` clause and/or `WHERE` clause
  - Nested query in FROM clause: conceptually evaluates and creates a temporary table

```
-- find the names of all the students who've taken CSE562
SELECT S.name
FROM students S,
     (SELECT sid FROM enrollment WHERE cno = 562) E
WHERE S.sid = E.sid;
```

  - Nested query in `WHERE` clause (actually also `HAVING` clause, see later)

```
SELECT name
FROM students
WHERE sid in (SELECT sid FROM enrollment WHERE cno = 562);
```

  - To find those who have not taken CSE562, use `NOT IN` operator

Nested Query (cont'd)

- Nested queries may also reference outer query relations
- Set operators in nested query
  - `EXISTS/NOT EXISTS`: whether the result of the subquery is non-empty/empty

```
SELECT name
FROM student S
WHERE EXISTS (SELECT * FROM enrollment E WHERE S.sid = E.sid AND cno = 562);
```

references outer query relation S

  - Set comparison `op SOME/ALL`: compares a value against a set (op is an operator such as <, <=, =, …)
    - `a > SOME (subquery)`: a is larger than some value in the result set of the subquery
    - `a > ALL (subquery)`: a is larger than all the values in the result set of the subquery

```
-- find the sid of all the students with the highest grade in CSE562
SELECT sid
FROM enrollment
WHERE cno = 562
  AND grade >= ALL (SELECT grade FROM enrollment
                    WHERE cno = 562 AND grade is not NULL);
```

Aggregation

- Aggregation operator is an extension to relational algebra
  - $\gamma_{F(expr),\ldots}Q$ where F is an aggregation function
  - Common aggregation function include:
    - COUNT(*) – number of result rows
    - COUNT(expr) – number of non-null rows
    - MIN, MAX, SUM, AVG, VARIANCE, STDDEV
  - Adding `DISTINCT` before the argument in the aggregation function
    - Deduplicate the expr values before aggregation
    - COUNT(DISTINCT *) *is not valid!*

```
SELECT  F([distinct] expr) [,…]
FROM    relation-list
[WHERE   predicate]
```

- Examples
  - `SELECT MAX(grade) FROM enrollment WHERE cno = 562` -- find the highest grade in CSE562
  - ```
    SELECT name from student where cno = 562
      AND grade = (SELECT MAX(grade) from enrollment where cno = 562)
    ```
    - find the names of the students who have the highest grade in CSE562
    ```

Aggregation with Grouping

- Can also have optional `GROUP BY` and `HAVING` clauses
  - `GROUP BY`: group the rows by distinct values of the expressions
    - expr can be any output column or any expression over input columns
    - target-list can have none/part/all of grouping exprs and any number of aggregation functions
    - aggregation functions are applied on a per-group basis
  - `HAVING`: a selection operator over the groups
    - can use any grouping expr or any aggregation function (not necessary in the target list)

```
SELECT   target-list
FROM     relation-list
[WHERE   predicate]
[GROUP BY expr1, expr2, …
[HAVING having-predicate]]
```

- In extended relational algebra:

$$\pi_{target-list}\sigma_{having-predicate}\left(\ {}_{expr1,expr2,\ldots}\gamma_{F(expr'_1),\ldots}Q\right)$$

where Q is the relational algebra for `SELECT * FROM relation-list WHERE predicate;`

Aggregation with Grouping (cont'd)

- Example 1: find the enrollment size of each 500-level or above courses
  - `SELECT semester, cno, COUNT(*) AS size FROM enrollment GROUP by semester, cno HAVING cno >= 500;`

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

result

semester	cno	size
s22	562	2
f21	560	3
s21	560	1

$$\sigma_{cno \geq 500} \left({}_{semester,cno}\gamma_{COUNT(*)\ as\ size} enrollment \right)$$

Aggregation with Grouping (cont'd)

- Example 2: find the enrollment size of all course with average GPA >= 3.0
  - `SELECT semester, cno, COUNT(*) AS size FROM enrollment GROUP by semester, cno HAVING AVG(grade) >= 3.0;`

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

result

semester	cno	size
f21	560	3
s21	560	1
f21	250	1

$$\pi_{semester,cno,size}\sigma_{avggpa\geq 3.0}\left({}_{semester,cno}\gamma_{COUNT(*)\ as\ size,AVG(grade)\ as\ avggpa}enrollment\right)$$

Null values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
  - SQL provides a special value *null* for such situations.
- The presence of *null* complicates many issues. E.g.:
  - Special operators needed to check if value `IS/IS NOT NULL`.
  - Is `rating>8` true or false when *rating* is equal to *null*? What about `AND`, `OR` and `NOT`?
  - We need a *3-valued logic* (true, false and *unknown*).
  - Meaning of constructs must be defined carefully.
    (e.g., WHERE clause eliminates rows that don't evaluate to true.)
  - New operators (in particular, *outer joins*) possible/needed.
- NULLs are usually ignored in aggregate functions
- Exercise: truth tables for OR and NOT operators?

Truth table for SQL AND

op1	op2	result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE
TRUE	NULL	NULL
FALSE	NULL	FALSE
NULL	NULL	NULL

Null values

- Seemingly "equivalent" queries may actually produce different results due to NULL values
  - e.g., find the sid of all the students with the highest grade in CSE562

```
SELECT sid
FROM enrollment
WHERE cno = 562
  AND grade = (SELECT MAX(grade) FROM enrollment WHERE cno = 562);
```

```
SELECT sid
FROM enrollment
WHERE cno = 562
  AND grade >= ALL (SELECT grade FROM enrollment
                    WHERE cno = 562);
```

Returns empty set if there's at least one NULL grade value in CSE562.
How to correct it?

Outer Join

- Explicit join semantics needed unless it is an INNER join

```
SELECT (column_list)
FROM  table_name
  [INNER | {LEFT |RIGHT | FULL } OUTER] JOIN table_name
    ON qualification_list
WHERE …
```

Set operations in SQL

- `INTERSECT`: ∩
- `UNION`: ∪
- `EXCEPT`: −

```
query1 INTERSECT [ALL] query2
query1 UNION [ALL] query2
query1 EXCEPT [ALL] query2
```

- Uses set semantics (i.e., deduplicate after the set operation)
  - unless `ALL` keyword is specified (i.e., no deduplication)

Other DML Statements

```
INSERT [INTO] table_name [(column_list)] VALUES ( value_list);
INSERT [INTO] table_name [(column_list)] <select statement>;

DELETE [FROM] table_name [WHERE qualification];
UPDATE SET column_name = expr [,…] [WHERE qualification];
```

Summary

- SQL review
  - DDL & DML
  - Multi-set relational algebra

- Next time: Physical Storage System