CSE462/562: Database Systems (Spring 24)

Lecture 7: Relational Model and SQL

3/11/2024

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

Data abstraction

- A revisit of the personal spending DB
- What if we want to
  - record the payment method
  - track budgets/bills
  - link entries to itemized receipts
- Or what if
  - the program/spreadsheet is slow after a while
  - you are managing the spending DB for many people (e.g., a company)

```
Date   Amount   Description
2/1    $20.21    Grocery
2/2    $10.54   Fast food
2/3    $39.22    Cell phone bill
…
2/27   $33.00    Clothes
```

- Constant changes in data management
  - for efficiency or for new application usages
  - impractical to break existing applications for every change

Data abstraction

- Data abstraction
  - View level: what and how to present data to different applications/users

    Logical Data Independence: ability to change logical schema without changing the external views and upper-level applications

  - Logical level: what data are stored

    Physical Data Independence: ability to change physical data storage without changing the logical schema

  - Physical level: how data are stored

Data models

- Data models are conceptual tools for
  - describing and defining the data abstractions
  - linking user's view to the bits stored in DBMS
- Many data models exist
  - Relational model (aka structured data model)
  - Entity-Relationship Model
  - Semi-structured data model
  - Graph data model
  - …

We'll focus on relational model and Relational DataBase Management Systems (RDBMS) in this course:

It's the foundation of many other data models (including semi-structured data model, graph data model and etc.).

- The survey below gives a historical view of why relational models are successful
  - Joseph M. Hellerstein and Michael Stonebraker. What Goes Around Comes Around. Readings in Database Systems, 4th Edition (2005).
  - Keep it simple and stupid!

Relational model

- Example: student records database

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

Relational model

- Relational database: a collection of named relations (aka tables)
- Relation: a set of records (aka tuples) – no duplicates
  - In reality: multi-set semantics are more prevalent – allow duplicates
- Record: a sequence of values
  - represents relationships among values

- Two concepts
  - Database Schema: names of the relations + names and types of the columns + constraints
    - e.g., student(sid: integer, name: string, login: string, major: string, adm_year: date)
    - each named column is also called an attribute or a field
  - Database instance: a snapshot of the data at a time point
    - e.g., the specific data in our student record database example

Relational model

Database schema

student(sid: integer, name: string, login: string, major: string, adm_year: date)

enrollment(sid: integer, semester: string, cno: integer, grade: float)

Relation (schema)

Relation (instance)

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

Record

Column

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

Database instance

Integrity constraints

- Key constraints
  - Superkey: a set of columns that uniquely identify a record
    - e.g., $\{sid\}$ is a superkey of student relation; $\{sid, name\}$ is too;
    - e.g., $\{sid, semester, cno\}$ is a superkey of enrollment relation; but $\{sid, cno\}$ is not
    - Has nothing to do with specific instances
      - $\{sid, cno\}$ is not a superkey even if no one's ever taken a course twice
      - but it will be if the university policy prohibits retaking the same course

  - Candidate key: a superkey $K\ s.t. \nexists K' \subset K: K'$ is a superkey
    - e.g., $\{sid\}$ and $\{login\}$ are both candidate keys of student; $\{sid, login\}$ is not

  - (Primary) key: a chosen candidate key by the database designer
    - e.g., student(sid: integer, name: string, login: string, major: string, adm_year: date)

Integrity constraints

- Foreign-key constraints
  - from attributes A of referencing relation R to primary key A' of referenced relation R':
  - such that for any DB instance, any value of A must appear in A' of some tuple in R'

R' = 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

R = 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

Integrity constraints

- Referential constraints
  - from attributes A of referencing relation R to *attributes* A' of referenced relation R'
  - such that for any DB instance, any value of A must appear in A' of some tuple in R'
  - Foreign-key constraints as a special case where A' is the primary key of R'

- Other general constraints

- These are less supported by DBMS due to efficiency reasons

Query Language

- Formal query languages
  - Relational algebra
    - Functional – describes how to query
  - Relational calculus
    - Declarative – describes what to query
  - No side effects! Does not include data definition, update, integrity checks, and etc.
  - Theoretical foundation of modern RDBMS; allows for query optimization

- Query language in practice: SQL (Structured Query Language)
  - Has its root in relational algebra and relational calculus
  - Includes many more beyond queries: imperative sublanguage, data definition, etc.

Relational algebra

- There are 6 basic operators:
  - Selection σ
  - Projection π
  - Renaming ρ
  - Cartesian product $\times$
  - Set difference $-$
  - Union $\cup$

- The operators takes relations as input, and outputs a relation
  - Schemas of the input/output schema are fixed
  - Operators can be composed

Selection

- $\sigma_P R$
  - Selects the records in relation R that satisfy a predicate P
  - Output relation has the same schema as its input

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

$$\sigma_{major='CS'} student$$

Projection

- $\pi_A R$
  - Retains only the attributes A in the output (i.e., "filters" on columns)
  - Schema of the result is exactly A
- Projection in relational algebra *must* eliminate duplicates
  - In practice, no for using multi-set relational algebra, unless requested by the user.

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

$\pi_{major,adm\_year} student$ →

major	adm_year
CS	2021
CE	2020
CS	2020

Renaming operator

- $\rho_{A_1 \to A_1', A_2 \to A_2', \ldots} R$
  - Renames the attributes $A_1, A_2, \ldots$ to $A_1', A_2', \ldots$
  - Output schema is same as R except that the attributes are renamed

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

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

$\rho_{login \to ubitname, 2 \to fullname} student$

Positional notation

Cartesian product

- $R_1 \times R_2$
  - Concatenates every pair of tuples $t_1 \in R_1, t_2 \in R_2$ into a single tuple $t \in R_1 \times R_2$
  - Output schema is the concatenation of the two input schemas
    - There might be naming conflicts, use renaming operator to avoid that

student

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

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7

$student \times enrollment$

sid	name	login	major	adm_year	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
101	Bob	bob5	CE	2020	100	s22	562	2.0
More results follows ……								

Union

- $R \cup R'$
  - Union of two relations of the *compatible* schema
  - Output schema remains the same as inputs

Same number of columns. The i^{th} columns in both relations have the same type for all i.

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020

new_students

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
102	Charlie	charlie7	CS	2021
104	Carol	carol20	CS	2021

$students \cup new\_students$

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
104	Carol	carol20	CS	2021

Set difference

- $R - R'$
  - Set difference of two relations of the *compatible* schema
  - Output schema remains the same as inputs

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020

new_students

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
102	Charlie	charlie7	CS	2021
104	Carol	carol20	CS	2021

$students - new\_students$

sid	name	login	major	adm_year
101	Bob	bob5	CE	2020

Assignment notation

- To help compose more complex queries with shared subqueries
  - $A \leftarrow Q$: A denotes the output of relational algebra expression Q
  - E.g.,

$$studentInCS \leftarrow \sigma_{major='CS'} student$$
$$students2021 \leftarrow \sigma_{adm\_year=2021} student$$
$$studentInCS \cup students2021$$

Compound operators

- Several useful compound operators
  - Join ⋈
    - Inner join
      - Natural join
    - Outer join
  - Set intersection ∩
  - Division operator /
  - …

- All of them can be composed from the 6 basic operators
- Does not add expressiveness of the relational algebra

Inner join

- $R \bowtie_P R' = \sigma_P(R \times R')$
  - Selecting records that satisfy the predicate P from $R \times R'$
- Most common special case is *natural join*

$$R \bowtie R' = \pi_{A(R) \cup A(R')} \sigma_{\forall a \in A(R) \cap A(R') : R.a = R'.a}(R \times R')$$

  - $A(R)$: attributes of R
  - The predicate P is implicitly equality between common attributes of R and R'
  - Projecting to all unique attributes of R and R' (only one copy for common attributes)
- Equi-join: P is conjunction of equality predicates
- Useful for denormalization

Natural join

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

$$S \bowtie E = \pi_{S.sid,S.name,S.login,S.major,S.adm_{year},E.semester,E.cno,E.grade}\sigma_{S.sid=E.sid}S \times E$$

sid	name	login	major	adm_year	semester	cno	grade
100	Alice	alicer34	CS	2021	s22	562	2.0
100	Alice	alicer34	CS	2021	f21	560	3.7
101	Bob	bob5	CE	2020	s21	560	3.3
102	Charlie	charlie7	CS	2020	s22	562	3.7

Inner join

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

sid	cno	grade	sid	cno	grade
100	562	2.0	102	562	2.3
100	560	3.7	102	560	4.0
101	560	3.3	100	560	3.7
101	560	3.3	102	560	4.0
100	560	3.7	102	560	4.0

$$E_1, E_2 \leftarrow \pi_{sid,cno,grade} E$$

$$E_1 \bowtie_{E_1.cno = E_2.cno \wedge E_1.grade < E_2.grade} E_2$$

Outer join

- Inner join results ∪ tuples without matches (augmented with NULLs)
- Types of outer joins
  - Left outer join $R ⟕ p\, R' = R \bowtie_P R' \cup \left(\left(R - \pi_{A(R)} R \bowtie_P R'\right) \times \{(\underbrace{\phi, \phi, \dots, \phi}_{|A(R')|\ NULLs})\}\right)$
  - Right outer join $R ⟖ p\, R' = R \bowtie_P R' \cup \left(\{(\underbrace{\phi, \phi, \dots \phi}_{|A(R)|\ NULLs})\} \times \left(R' - \pi_{A(R')} R \bowtie_P R'\right)\right)$
  - Full outer join

$$R ⟗ p R' = R ⟕ p\, R' \cup R ⟖ p\, R'$$

- Useful for preserving all unique values in one or both relations

Outer join

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

$S ⟕_{S.sid=E.sid} E$

S.sid	S.name	S.login	S.major	S.adm_year	E.sid	E.semester	E.cno	E.grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
100	Alice	alicer34	CS	2021	100	f21	560	3.7
101	Bob	bob5	CE	2020	101	s21	560	3.3
102	Charlie	charlie7	CS	2020	102	s22	562	2.3
103	David	davel	CS	2020	NULL	NULL	NULL	NULL

Other useful operators

- Set intersection: $R \cap R' = R - (R - R')$

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020

new_students

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
102	Charlie	charlie7	CS	2021
104	Carol	carol20	CS	2021

$students \cap new\_students$

sid	name	login	major	adm_year
100	Alic	alicer34	CS	2021

Other useful operators

- Division: R/R'
  - Attributes of R' must be a subset of the attributes of R
  - The output schema of division is the extra attributes $A_o = A(R) - A(R')$ of R
  - R/R' contains all tuples $t_o \in \pi_{A_o} R$ such that for every $t' \in R'$, the concatenation $t_o \circ t' \in R$
- Useful for expressing "for all" queries like
  - Find all students who have enrolled in both CSE560 and CSE562

Division

Find all students who have enrolled in both CSE560 and CSE562

E

sid	cno
100	562
102	562
100	560
101	560
102	560
103	460
101	560
103	250

C

cno
562
560

E / C

sid
100
102

Division

- Exercise: how to express R/R' using the basic operators
- Idea: find all $t_o \in \pi_{A_o} R$ such that some combination $t_o \circ t'$ is missing from R
- $R/R' = \pi_{A_o} R - \pi_{A_o}\left(\left(\pi_{A_o} R \times R'\right) - R\right)$

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]}
```

can only reference multiple table column's values

where *action* is one of:

```
NO ACTION, CASCADE, SET NULL, SET DEFAULT
```
```

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 $R_1, R_2, \dots, R_n$ be relations in the relation list
  and $E_1, E_2, \dots, E_m$ be the expressions in the target list
  and P be the boolean predicate in the WHERE clause (P = true if WHERE clause is missing)

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

- 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;
```

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

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

$S \times E$

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

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

- Exercise: truth tables for OR and NOT operators?

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

- Relational model, relational algebra & SQL

- Next lecture: query processing overview

- Reminders
  - No office hours during the Spring Recess (3/18/2024 - 3/23/2024)
    - Post questions on Piazza + mid-term review + Q&A on 3/25
  - HW3 due this Sunday (3/17/2024, 23:59 PM EDT)
  - Project 3 due next Sunday (3/24/2024, 23:59 PM EDT)
  - Midterm exam on 3/27/2024, Knox 104, 7:05 pm - 8:25 pm
    - Open-book, paper materials only, no electronics except a calculator