SFU

Database Systems I

CMPT 354 Summer 2024

Zhengjie Miao

10 July 2024

All topics at a glance

Data scientist	Full-stack developer	DB admin/backend developer	Data platform engineer	Fair game	Emphasized	
						Structured data
☑	☑	☑	☑	☑	☑	• Relational model & algebra
	☑	☑				• E/R design
	☑	☑	☑	☑		• Relational design theory (redundancies/dependencies, normal forms)
☑	☑	☑	☑	☑	☑	• SQL model (bag vs. set, NULL)
☑	☑	☑	☑	☑	☑	• SQL querying & modification (grouping/aggregation, subqueries…)
	☑	☑	☑	☑	☑	• SQL constraints & triggers
	☑	☑	☑	☑	☑	• SQL transactions (ACID)
☑	☑	☑	☑	☑	☑	• SQL + programming (cursor, injection attack, prepared statements, full-stack…)
						Semi-structured data
☑	☑	☑	☑	☑	☑	• XML & XPath (XQuery ☑)
☑	☑	☑	☑	☑	☑	• JSON & MongoDB pipelines (analogous to relational algebra)
						Database internals
☑	☑	☑	☑	☑	☑	• Storage (memory hierarchy, trips to Pluto)

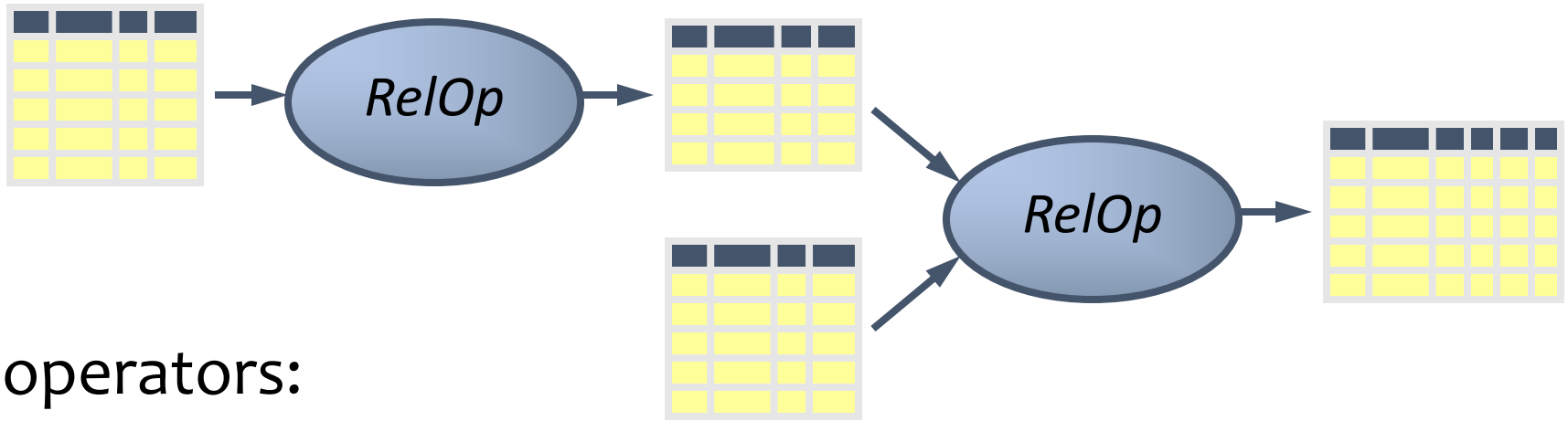
Fair game:
may be in multiple choice

Emphasized:
will be in long answer question

Be able to track the time an operation takes in a database

Relational algebra

A language for querying relational data based on “operators”



- Core operators:
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, natural join, intersection, etc.
- Compose operators to make complex queries

Why is "–" needed for "highest"?

- Composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input
- Is the "highest" query monotone?
  - No!
  - Current highest *pop* is 0.9
  - Add another row with *pop* 0.91
  - Old answer is invalidated
- So it must use difference!

- Exam I P2f.
- Track for each question the top-1 leader whose correct query runs the fastest

$$\pi_{\text{question\_id, user\_id, runtime}}(\text{Queries}) -$$
$$\pi_{\text{q1.question\_id, q1.user\_id, q1.runtime}}($$
$$\rho_{\text{q1}}(\text{Queries}) \bowtie_{\text{q1.question\_id=q2.question\_id AND q1.runtime >q2.runtime}} \rho_{\text{q2}}(\text{Queries}))$$

- Also Exam I P4d

Keys

- A set of attributes K is a key for a relation R if
  - In no instance of R will two different tuples agree on all attributes of K
    - That is, K can serve as a "tuple identifier"
  - No proper subset of K satisfies the above condition
    - That is, K is minimal
- Example: *User* (*uid*, *name*, *age*, *pop*)
  - *uid* is a key of *User*
  - *age* is not a key (not an identifier)
  - {*uid*, *name*} is not a key (not minimal)

Redefining "keys" using FD's

A set of attributes K is a key for a relation R if

- $K \rightarrow$ all (other) attributes of R
  - That is, K is a "super key"
- No proper subset of K satisfies the above condition
  - That is, K is minimal

- Exam I P1d.
- All attributes of a relation are always a super key of that relation.

- Exam I P1j.
- Consider relation $R(A, B, C, D)$. Suppose we know $\{A, B\}$ is a key of R (and R may or may not have other keys). Then the FD $C \rightarrow AB$ cannot hold in R.

SQL: Set versus bag semantics

- Set
  - No duplicates
  - Relational model and algebra use set semantics
- Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - SQL uses bag semantics by default

SQL set and bag operations

- `UNION, EXCEPT, INTERSECT`
  - Set semantics
    - Duplicates in input tables, if any, are first eliminated
    - Duplicates in result are also eliminated (for UNION)
  - Exactly like set ∪, −, and ∩ in relational algebra
- `UNION ALL, EXCEPT ALL, INTERSECT ALL`
  - Bag semantics
  - Think of each row as having an implicit count (the number of times it appears in the table)
  - Bag union: sum up the counts from two tables
  - Bag difference: proper-subtract the two counts
  - Bag intersection: take the minimum of the two counts

Exam I P1g & P1h

Given relations R and S (with possible duplicates), if R has m copies of 1 and S has n copies of 1, then **for all values** of m and n such that **m >= n**:
The following queries are equivalent, where ⊎, ⋒, and − denote bag union, bag intersection, and bag difference

- $R \uplus S$
- $R - ((R - S) \uplus (S - R))$

Quantified subqueries

- A quantified subquery can be used syntactically as a value in a WHERE condition
- Universal quantification (for all):
  ... WHERE x *op* ALL($subquery$) ...
  - True iff for all t in the result of $subquery$, x *op* t
- Existential quantification (exists):
  ... WHERE x *op* ANY | SOME($subquery$) ...
  - True iff there exists some t in $subquery$ result such that x *op* t
  - Beware
    - In common parlance, “any” and “all” seem to be synonyms
    - But in SQL, ANY really means “some”

- Exam I P2i.
- Track for each question the top-1 leader whose correct query runs the fastest

```
SELECT question_id, user_id, runtime FROM Queries q
WHERE runtime <= ANY(SELECT runtime FROM Queries
                     WHERE question_id = q.question_id);
```

Scalar subqueries

- A query that returns a single row can be used as a value in WHERE, SELECT, etc.
- Example: users at the same age as Bart
  - ```
    SELECT *
    FROM User
    WHERE age = (SELECT age
                 FROM User
                 WHERE name = 'Bart');
    ```
  - Runtime error if subquery returns more than one row
    - Under what condition will this error never occur?
  - What if the subquery returns no rows?
    - The answer is treated as a special value NULL, and the comparison with NULL will fail

What's Bart's age?
    ```

Computing with NULL's

- When we operate on a NULL and another value (including another NULL) using +, −, etc., the result is NULL
- Aggregate functions **ignore** NULL, except COUNT(*) (since it counts rows)
- Evaluating aggregation functions (except COUNT) on an empty collection returns NULL; converting an empty collection to a scalar also gives NULL

Exam I P4e. Below is an SQL query to find all students (sid, sname) who ranked in top-3 in more contests where their department is the host than other departments are the host:

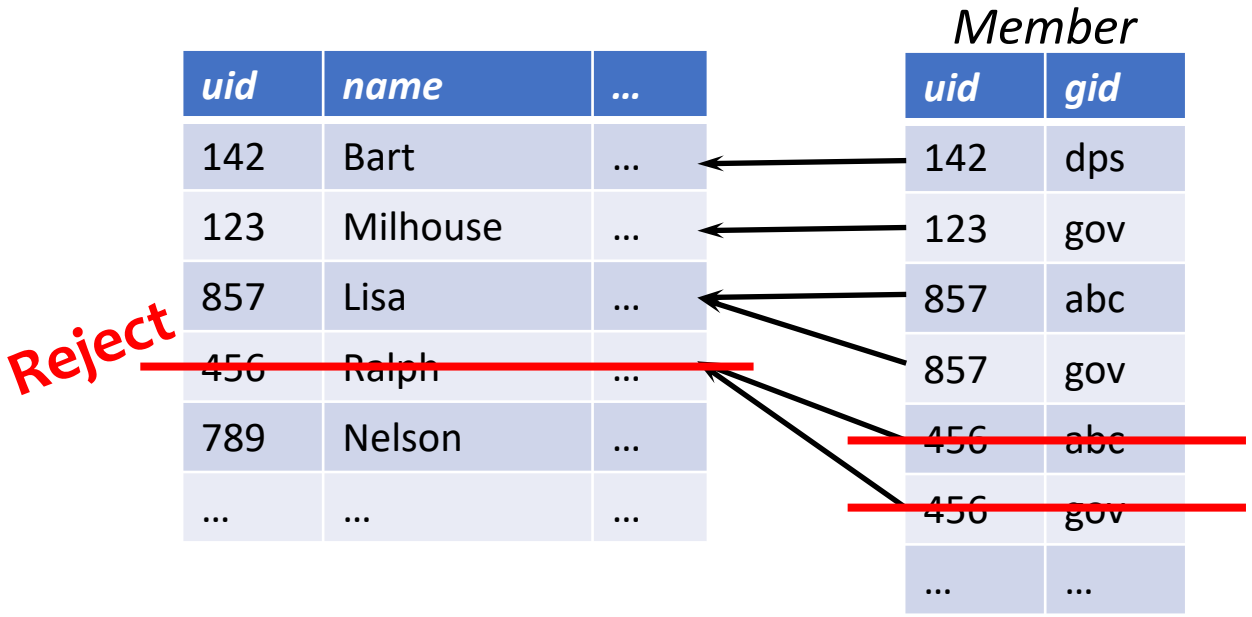
```
WITH SPC AS (
    SELECT p.sid, s.sname, c.cid, s.dept, c.host_dept, p.rank
    FROM Student s
    JOIN Participate p ON s.sid = p.sid JOIN Contest c ON p.cid = c.cid
    WHERE p.rank in (1,2,3)
)
SELECT r1.sid, r1.sname
FROM SPC AS r1
WHERE r1.dept = r1.host_dept
GROUP BY r1.sid, r1.sname
HAVING COUNT(*) > (
    SELECT COUNT(*) FROM SPC AS r2
    WHERE r1.sid = r2.sid AND r2.dept <> r2.host_dept
);
```

- Old answer: Students who ranked in top-3 in contests hosted by their departments but never ranked in top-3 in contests hosted by other departments will be missed. Use coalesce in the scalar subquery in HAVING count(*) > ()…
- The point was that aggregate functions on an empty input return NULL, hence the HAVING count(*) > NULL would not work; however, COUNT is an exception that would return 0 (e.g., if the query used AVG(rank), the AVG value would be NULL for those students who were supposed to be "missing" in the original answer). Also, see the doc (https://www.postgresql.org/docs/14/functions-aggregate.html).

Enforcing referential integrity

Example: *Member.uid* references *User.uid*

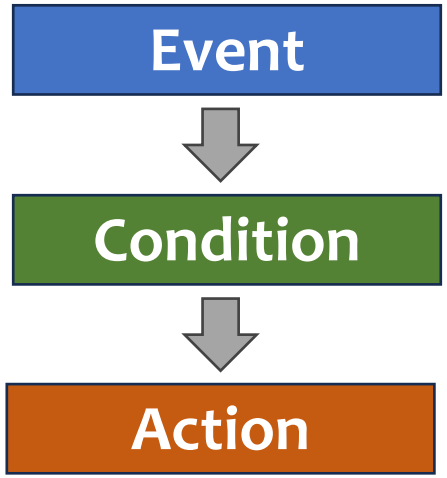
- Delete or update a *User* row whose *uid* is referenced by some *Member* row
  - Option 1: Reject
  - Option 2: Cascade --- ripple changes to all referring rows




```
CREATE TABLE Member
(uid INT NOT NULL REFERENCES
User(uid) ON DELETE CASCADE,
…..);
```

Triggers

- A trigger is an event-condition-action (ECA) rule
  - When event occurs, test condition; if condition is satisfied, execute action
  - Different DBMS support different syntax, but concepts remain the same
    - E.g., PostgreSQL syntax ≠ what we'll present here



Delete/update a row in User

Whether its uid is referenced by some row in Member

If Yes: reject/delete/ cascade/NULL

Referential constraints

Some user's popularity is updated

The user is a member of cks ("Cool Kids") and *pop* drops below 0.5

If Yes: kick that user out of cks

Data Monitoring

- Exam II Sample Questions
- P3b. (skip)

- P3c. Assume there is a trigger on Contest and a trigger on Participate (not the one in 3b), can an UPDATE statement on Contest cause the trigger on Participate to fire? (Y or N)
- Answer: Y
- The trigger on Contest's action may modify Participate.

Transactions

- A transaction is a sequence of database operations with the following properties (ACID):
  - Atomic: Operations of a transaction are executed all-or-nothing, and are never left "half-done"
  - Consistency: Assume all database constraints are satisfied at the start of a transaction, they should remain satisfied at the end of the transaction
  - Isolation: Transactions must behave as if they were executed in complete isolation from each other
  - Durability: If the DBMS crashes after a transaction commits, all effects of the transaction must remain in the database when DBMS comes back up

- Exam II Sample Questions

1a. (T/F) In a program, multiple statements can be grouped together as a transaction.
Answer:
True

1b. (T/F) The actions in a transaction are atomic and either they are all performed or none of them are performed.
Answer:
True

Simple XPath examples

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>…
</bibliography>
```

- All book titles
  `/bibliography/book/title`
- All book ISBN numbers
  `/bibliography/book/@ISBN`
- All title elements, anywhere in the document
  `//title`
- All section titles, anywhere in the document
  `//section/title`
- Authors of bibliographical entries (suppose there are articles, reports, etc. in addition to books)
  `/bibliography/*/author`

More complex predicates

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>…
</bibliography>
```

Predicates can use and, or, and not

- Books with price between $40 and $50

  `/bibliography/book[40<=@price and @price<=50]`

- Books authored by "Abiteboul" or those with price no lower than $50

  `/bibliography/book[author='Abiteboul' or @price>=50]`

  `/bibliography/book[author='Abiteboul' or not(@price<50)]`

  - Any difference between these two queries?

- # Exam II Sample Questions

1c. (T/F) Consider the following two XPath queries:

- `//A[B/C = "foo" and B/D = "bar"]`
- `//A[B[C = "foo" and D = "bar"]]`

For the first query, you only need C and D to be within any B element

Every element returned by the first query will be returned by the second.

Answer:

For the second query, you would need C and D to be within the B element

False. Consider the following XML document:

```
<A>
<B><C>foo</C></B>
<B><D>bar</D></B>
</A>
```

The first query will return this element; the second query will not.

1d. (T/F) consider the XPath queries above, every element returned by the second query will be returned by the first.

Answer:

True. The C and D elements that make an A element satisfy the condition in the second query will make the same A element satisfy the condition in the first query

- Exam II Sample Questions

P2a. Write an XPath expression that are equivalent to the XQuery below.

```
  for $c in /Registration/Course
  return
    if (exists($c/Student[Grade >= 90 and Grade < 95])) then
$c/Number
```

Answer:

```
/Registration/Course[Student[Grade >= 90 and Grade < 95]]/Number
```

Or

```
/Registration/Course[Student[Grade >= 90][Grade < 95]]/Number
```

Or

```
/Registration/Course[count(./Student[Grade >= 90 and Grade < 95]) >
0]/Number
```

Note that `/Registration/Course[./Student/Grade >= 90 and ./Student/Grade < 95]/Number` is wrong because there can be multiple students in a course, and the condition checks if at least one student's grade is >= 90 and at least one student's grade is < 95.

MongoDB: No general "twig" matching!

- Suppose for a moment `publisher` is an object itself, with attributes `name`, `state`, and `country`
- The following query won't get you database books by US publishers:

```
db.bib.find({ title: /[dD]atabase/,
              publisher: { country: "US" } })
```

  - Instead, the condition on `publisher` is satisfied only if it is an object with exactly one attribute, and this attribute must be named `country` and has value `"US"`
  - What happens is that MongoDB checks the equality against `{country: "US"}` *as an object*, not as a pattern!

Array operators vs. unnest/nest

Don't array operators `$map`/`$filter` *look like projection/selection to you?*

- You can always unnest, project/select, and then nest (aggregate) them back!
- In nested relational algebra, which could serve as the theoretical foundation for querying nested data, you just need the following operators:
  - $\cup, \cap, -, \sigma_p, \pi_L$
  - `tup_create/destroy` (which enable $\times$ and $\rho_{\ldots}$)
  - `set_create/destroy` (which further enable `unnest`)
  - `nest`

(Interestingly, a more expressive language can be obtained by replacing `nest` *by* `powerset`*, which would allow you to express transitive closure!)*

- Exam II Sample Questions

P2c. Complete the MongoDB query below to retrieve the students whose grade is above the average grade for each course. Each output object has three fields, course number, student name, and student grade.

```
db.collection.aggregate([
  {
    $unwind: "$students"
  },
  { // Group by course and calculate the average grade
    $group: {
      _id: "$number",
      averageGrade: {
        $avg: "$students.grade"
      },
      students: {
        $push: "$students"
      }
    }
  },
  {
    $unwind: "$students"
  },
  { // compare student grade with the course average, $gt is >
    $match: {
      $expr: { $gt: ["$students.grade", "$averageGrade"] }
    }
  },
  {
    $project: {
      _id: 0, course: "$_id", studentName: "$students.name", studentGrade:
"$students.grade"
    }
  }
])
```