Session 3

Relational Queries

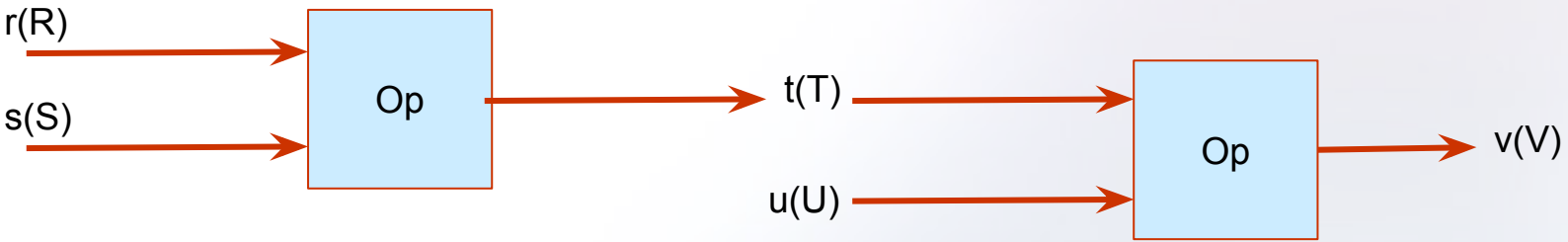
sourcemind

Relational queries

- A query language (QL) is a language that allows users to manipulate and retrieve data from a database.
- SQL is based on Relational Algebra
- Sample queries
  - Find the products sold by “Gadgets World”
  - Find the price of an “Apple Watch”
  - Find the website of the seller of an “Apple Watch”
- Query Language is the syntax that allows us to express these queries in our system and get them executed

Relational operations

- Query languages are composed of operations
- Operations that take one or two relations as input
- They always produce a single operation as the output
- Allow for using the output of an operation as an input of next operation, thus enabling chaining of operations



Relational algebra

- Queries in relational algebra are applied to relation instances, result of a query is again a relation instance
- Six basic operators in relational algebra:

Operator	Symbol	Description
Selecion	σ	selects a subset of tuples from a relation
Projection	π	deletes unwanted columns from a relation
Cartesian product	×	allows to combine two relations
Set difference	−	tuples in relation 1, but not in relation 2
Union	∪	tuples in relation 1 plus tuples in relation 2
Rename	ρ	renames attribute(s) and relation

sourcemind

Selection: $\sigma_P(r)$

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- Given a relation r with schema R, select only the tuples where the condition **P** holds over **r** (is true)
- P is a simple or complex boolean expression
- Syntax

$$\sigma_{A=B \ \wedge \ D > 5}(r)$$

A	B	C	D
α	α	1	7
β	β	23	10

- General form:

$$\sigma_P(r) := \{t \mid t \in r \text{ and } P(t)\}$$

sourcemind

Declarative queries

- Relational query languages are **declarative**, meaning we don't specify the steps to execute.
- We specify the **properties** of the result.
- The **query engine** (DBMS) must figure out how to execute the queries.

```
List list = new List()
for ( Row t : r ) {
    if (t.A == t.B && t.D > 5) {
        list.add( t )
    }
}
return list
```

$$\sigma_{A=B \wedge D > 5}(r)$$

Exercise

Write queries to:

Select products more expensive than 100 USD

Select products more expensive than 40,000 Drams

* All prices in the Product table are in USD

ID	Title	Description	Price	SellerID
123	iPod	Blah blah	100	1
201	Apple Watch	Blah blah	200	2
202	Charger	Blah blah	10	1
312	Wallet	From leather, black	30	3
124	iPad	iPad mini, white	400	1

Projection: $\prod_A(r)$

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

- Given a relation r with schema R with attributes A1, A2, ..., An, the projection over relation r and attributes A1, A2, ..., Ak is the relation that has k columns obtained by erasing all columns from r that are not listed.
- Duplicate rows are removed from result because relations are sets.
- Syntax

$\pi_{A1,A2,...,Ak}(r)$

Exercise

- Selection product title
- Selection seller IDs

ID	Title	Description	Price	SellerID
123	iPod	Blah blah	100	1
201	Apple Watch	Blah blah	200	2
202	Charge r	Blah blah	10	1
312	Wallet	From leather, black	30	3
124	iPad	iPad mini, white	400	1

Cartesian product: r × s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- Defined as: r × s := {tq | t ∈ r and q ∈ s}
- The combination of all rows from both tables
- Assume that attributes of r(R) and s(S) are disjoint, i.e., R ∩ S = ∅.

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian product with non-disjoint attributes

- If attributes of r(R) and s(S) are not disjoint, then the rename operation must be applied first.
- Column B in relation r → r.B
- Column B in relation s → s.B

A	B
α	1
β	2

r

B	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

A	r.B	s.B	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Exercise: Compute I × D

Relation I = (inst_id, inst_name, bank_acct, dept_name, salary)

inst_id	inst_name	bank_acct	dept_name	salary
1	Alex	182353	Computer Science	10000
3	Winston	893036	Political Science	10000
5	Sarah	638303	Law	8000

Relation D = (dept_name, dean_id, building, floor)

dept_name	dean_id	building	floor
Computer Science	1	N2	3
Political Science	3	N2	2
Music	NULL	N1	1

I × D

inst_id	inst_name	bank_acct	I.dept_name	salary	D.dept_name	dean_id	building	floor
1	Alex	182353	Computer Science	10000	Computer Science	1	N2	3
1	Alex	182353	Computer Science	10000	Political Science	3	N2	2
1	Alex	182353	Computer Science	10000	Music	NULL	N1	1
3	Winston	893036	Political Science	10000	Computer Science	1	N2	3
3	Winston	893036	Political Science	10000	Political Science	3	N2	2
3	Winston	893036	Political Science	10000	Music	NULL	N1	1
5	Sarah	638303	Law	8000	Computer Science	1	N2	3
5	Sarah	638303	Law	8000	Political Science	3	N2	2
5	Sarah	638303	Law	8000	Music	NULL	N1	1

Exercise: Find Alex's address

1. Compute the cartesian product of I and D (`I × D`)
2. Select only the rows where `I.dept_name` and `D.dept_name` match
3. Select the row by instructor's name `inst_name`
4. Project the result to keep only the instructor's name, building name, and floor number

Solution

$$r = I \times D$$

$$t = \sigma_{I.dept\_name=D.dept\_name \ \wedge \ inst\_name='Alex'}(r)$$

$$\Pi_{inst\_name,building,floor}\ (t)$$

$$\Pi_{inst\_name,building,floor}\ (\sigma_{I.dept\_name=D.dept\_name \ \wedge \ inst\_name='Alex'}(I \times D))$$

sourcemind

Since the result of relational operations is also a relation, we can **chain** the operations and make complex queries.

Set Difference: r – s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
β	1

- Defined as: r – s := {t | t ∈ r and t ∉ s}
- For r – s to be computable:
  1. r and s must have the same dimension (number of attributes)
  2. Attribute domains must be compatible

Union: r ∪ s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
α	2
β	1
β	3

- Defined as: Defined as r ∪ s := {t | t ∈ r or t ∈ s}
- For r ∪ s to be computable:
  1. r and s must have the same dimension (number of attributes)
  2. Attribute domains must be compatible

Natural Join: r ⋈ s

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

- The natural join of relations r and s, r ⋈ s, is a combined operation:
  1. Cartesian product of r and s, r × s
  2. Selection based on **equality** of all **common attributes** of r and s
  3. Projection based on union of attributes of r and s

$$r \bowtie s = \Pi_{A,\, r.B,\, C,\, r.D,\, E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$$

sourcemind

Exercise: Compute I ⋈ D

Relation I = (inst_id, inst_name, bank_acct, dept_name, salary)

inst_id	inst_name	bank_acct	dept_name	salary
1	Alex	182353	Computer Science	10000
3	Winston	893036	Political Science	10000
5	Sarah	638303	Law	8000

Relation D = (dept_name, dean_id, building, floor)

dept_name	dean_id	building	floor
Computer Science	1	N2	3
Political Science	3	N2	2
Music	NULL	N1	1

Summary of relational algebra operations

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary} >= 85000}(\text{instructor})$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(\text{instructor})$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
x (Cartesian Product)	*instructor* x *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
∪ (Union)	$\Pi_{name}(\text{instructor}) \cup \Pi_{name}(\text{student})$
	Output the union of tuples from the *two* input relations.
– (Set Difference)	$\Pi_{name}(\text{instructor}) - \Pi_{name}(\text{student})$
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

sourcemind

Rename $\rho_x(r)$

- It is sometimes useful to rename a relation
  - To assign a name to a calculated result
  - To assign a new name to an existing relation
- Also known as **copy** operation

Example:

To compute cartesian product of a relation with itself

Relations *r*

A	*B*
α	1
β	2

r

$r \times \rho_s(r)$

r.A	*r.B*	*s.A*	*s.B*
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

sourcemind

Homework 3

1. Using the Instructor and Department relations (tables) in the examples, write relational queries to compute:
   a. List of bank accounts in the Law department
   b. List of departments and their corresponding deans' names
      R (dept_name, inst_name),
      where in each row, the instructor with `inst_name` is the dean of departname with `dept_name`

2. Given a relation over R(value), using only the relational algebra, find the maximum value in the relation.

   Example `r = { (4), (1), (2), (5), (3), (6) }`

SQL by example: A library

Library tables

`Book (id, title, author)`

Later add:

`Author (id, name)`

`Topic (id, name)`

SQL by example: A library

During this exercise we will use SQL to build a database of books (library) in our Postgres instance.

We will then add data for sample books and search for books by different criteria.

Database design steps:

1. Entity-Relationship (ER) design
2. Converting to relational model
3. Implementing the relational model in a database (Postgres, MySQL, Oracle, etc.)

Creating and populating tables

- Creating tables
  - Data types
  - Primary keys
  - Constraints
- Populating tables

```
CREATE TABLE TableName (
    field1      datatype,
    field2      datatype,
    PRIMARY KEY (field1),
    FOREIGN KEY field2 REFERENCES AnotherTable(field)
);
```

CREATE TABLE

```
CREATE SCHEMA `library`;
SET search_path TO `library`;

CREATE TABLE book (
    id    SERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    author VARCHAR(100) NULL
);
```

```
INSERT INTO book VALUES (1, 'Databases');
INSERT INTO book(title, author) VALUES
('Java', 'James Gosling');
INSERT INTO book(title, author) VALUES
('MYSQL', NULL), ('ORACLE', 'Oracle
Publishing');
INSERT INTO book(title) VALUES ('C++')
RETURNING id;

SELECT * FROM book;
DELETE FROM book;
SELECT * FROM book;
```

Auto-generated keys in Postgres

- You want to generate primary keys automatically: 1, 2, 3, ... to avoid duplicates
- There are two methods to use auto-generated keys in Postgres

1) Using the `SERIAL / BIGSERIAL` data type
2) Using a custom sequence

```
CREATE SEQUENCE book_sequence start 1 increment 1;

CREATE TABLE book (
    id    INTEGER PRIMARY KEY DEFAULT nextval('book_sequence'),
    title VARCHAR(100) NOT NULL,
    author VARCHAR(100) NULL
);
```

INSERT using sequence

- Either provide a value, or let the DEFAULT pick `nextval(...)` from the sequence.

```
INSERT INTO book VALUES (nextval('book_sequence'), 'Databases');

INSERT INTO book(title, author) VALUES ('Java', 'Brian Goetz');

INSERT INTO book(title, author) VALUES
     ('MYSQL', NULL), ('ORACLE', 'Oracle Publishing');

INSERT INTO book(title) VALUES ('C++') RETURNING id;

SELECT * FROM book;
```

SELECT

- Select all books
- Select books by title
- Select books and order by title
  - Using field name
  - Using field number
  - Ascending and descending

```
SELECT * FROM <table> WHERE <condition> ORDER BY <column>;
```

ALTER Table

- Add a topic field to the book table
- Add a topic table and a foreign key to the book table
- Add multiple topics to a book

```
ALTER TABLE book
    ADD COLUMN topic VARCHAR(50) NOT NULL DEFAULT 'N/A';
```

Foreign Key constraint

```
CREATE TABLE topic (
    id SERIAL PRIMARY KEY
    -- Choose other columns and types
);

ALTER TABLE book ADD COLUMN topic_id INT;
ALTER TABLE book ADD CONSTRAINT fk_book_topic FOREIGN KEY (topic_id) REFERENCES
topic (id);

INSERT INTO book(title, author, topic_id) VALUES ('Testing', 'John Smith', 1);
```

NULL

- A **marker** that represents **missing** value
- Create the author table with `SERIAL id, VARCHAR(50) name, VARCHAR(50) email`
- Choose name as not nullable and email as nullable
- Populate with some tuples
- Try comparing
  - By “=”
  - By “IS NULL”

```
1)  SELECT name, email FROM author WHERE email = '';

2)  SELECT name, email FROM author WHERE email = NULL;

3)  SELECT name, email FROM author WHERE email IS NULL:
```

Try these commands

```
CREATE TEMP TABLE temp_author AS SELECT name FROM author;
SELECT * FROM temp_author;

UPDATE temp_author SET name = 'Bob 2' WHERE name = 'Bob';

DELETE FROM temp_author;

INSERT INTO temp_author SELECT * FROM author;

TRUNCATE temp_author;
```

LIKE and ILIKE

- LIKE operator matches string using wildcards

```
SELECT * FROM author WHERE name LIKE 'A%';

SELECT * FROM author WHERE name LIKE '%A%';

SELECT * FROM author WHERE upper(name) LIKE '%O%';

SELECT * FROM author WHERE name ILIKE '%a%';
```

COALESCE operator

- Returns the first non-null value from a set
- Used for substituting a null value in a query with a default value

```
SELECT title, author FROM book;
SELECT title, coalesce(author, 'No Author') AS author FROM book;
```

DISTINCT operator

- Removes duplicate tuples (rows) from a relation (table)
- Example
  Get all unique authors from book table
- Internally works by sorting tuples so it is slower than regular SELECT

```
SELECT coalesce(author, 'No Author') AS author FROM book;
SELECT DISTINCT coalesce(author, 'No Author') AS author FROM book;
```

LIMIT and OFFSET

- `LIMIT n` includes only n tuples in the result
- `LIMIT n OFFSET m` includes only n tuples in the result, starting from the m-th tuple
- One usage of `LIMIT 0` is to copy relation structure without the tuples

```
SELECT * FROM book LIMIT 2;
SELECT * FROM book LIMIT 2 OFFSET 1;
CREATE TABLE book_copy AS SELECT * FROM book LIMIT 0;
```

Viewing Postgres table schema (psql)

- Connect `psql postgres://localhost:5433/postgres -U postgres -W`
- Describe using `\d <schema name>.<table name>`

```
vahepezeshkian@Vahes-MacBook-Pro-2 dev %
vahepezeshkian@Vahes-MacBook-Pro-2 dev % psql postgres://localhost:5433/postgres -U postgres -W
Password:
psql (13.2, server 13.4 (Debian 13.4-1.pgdg100+1))
Type "help" for help.

postgres=# \d library.book
                                      Table "library.book"
  Column   |          Type          | Collation | Nullable |                   Default
-----------+------------------------+-----------+----------+-----------------------------------------
 id        | integer                |           | not null | nextval('library.book_sequence'::regclass)
 title     | character varying(100) |           | not null |
 author    | character varying(100) |           |          |
 topic     | character varying(50)  |           | not null | 'N/A'::character varying
 topic_id | integer                |           |          |
Indexes:
    "book_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "fk_book_topic" FOREIGN KEY (topic_id) REFERENCES library.topic(id)
```

sourcemind

Viewing Postgres table schema (SQL)

- `information_schema` contains tables about tables (meta-data)

```
SELECT table_name,
       column_name,
       data_type
FROM information_schema.columns
WHERE table_name = 'book';
```

```
SELECT DISTINCT data_type FROM information_schema.columns;
```

Subqueries (nested queries)

- Recall that relational algebra operators accept relations and produce relations
- In SQL also we can chain queries
  - IN / NOT IN
  - EXISTS / NOT EXISTS
  - Different types of JOIN

IN / NOT IN

- Check attributes against a relation
  - A set of tuples
  - Result of a query

```
SELECT * FROM book WHERE topic IN ('Databases', 'Postgres', 'MySQL');
```

```
SELECT *
FROM book b
WHERE b.author IN
(SELECT name FROM author a WHERE lower(a.name) IN ('bob', 'alice'));
```

EXISTS / NOT EXISTS

- Returns true if the provided query is not empty
- Returns false if the provided query is empty

```
SELECT *
FROM book b
WHERE EXISTS(SELECT 1);
```

```
SELECT *
FROM book b
WHERE EXISTS
    (SELECT * FROM author
        WHERE b.author =   author.name
AND author.active = 1);
```

Exercise

- Create a table with only one column containing numbers
- Insert some numbers into the table
- Using nested queries find the maximum number in the table

JOIN

- Recall cartesian product and natural join from relational algebra
- Prepare data in `book`, `topic`, `author` tables

- SQL joins work the same as relational algebra
  - Cartesian product
  - Natural join
  - Inner join
  - Left, right, and full outer join

Prepare some data...

- Add `topic_name` to table `topic`
- Remove duplicate `topic` column from `book`
- Add some topics
- Add some books linked to the topics

Prepare some data (script)

```
SELECT * FROM topic;
ALTER TABLE topic ADD COLUMN topic_name VARCHAR(100);

SELECT * FROM book;
TRUNCATE TABLE book;
ALTER TABLE book DROP COLUMN topic;

INSERT INTO topic(topic_name)
VALUES ('Databases'),
       ('Programming'),
       ('Network'),
       ('Operating Systems');

INSERT INTO book(title, author, topic_id)
VALUES ('MySQL in Action', null, 1),
       ('Pro Postgres', null, 1),
       ('Java Core', null, 2);

CREATE TABLE topic_copy(topic_id, topic_name) AS SELECT * FROM topic;
```

JOINs

- Cartesian product: book × topic mixes all the tuples
- Natural join: book ⋈ topic mixes tuples with matching column names and values
- Inner join: mixes tuples with a given condition

```
SELECT * FROM book, topic;
SELECT * FROM book NATURAL JOIN topic;
SELECT * FROM book NATURAL JOIN topic_copy;
SELECT * FROM book INNER JOIN topic on book.topic_id = topic.id;
```

Outer joins

- Right outer join
- Left outer join
- Full outer join

```
SELECT * FROM book RIGHT OUTER JOIN topic on book.topic_id = topic.id;

SELECT * FROM book LEFT OUTER JOIN topic on book.topic_id = topic.id;

SELECT * FROM book FULL OUTER JOIN topic on book.topic_id = topic.id;
```