Introduction to SQL Data Manipulation Language (DML)

CSCI 220: Database Management and Systems Design

Today you will learn…

- How the relational calculus and relational algebra are used in real-world databases
- How to retrieve data using SQL

Previously: Database Changes

- You learned how to a relational schema using the SQL DDL (Structured Query Language Data Definition Language)

- Create the Loan table:
  ```
  CREATE TABLE loan (id INTEGER, amount MONEY)
  PRIMARY KEY (id);
  ```

- Insert into the Loan table:
  ```
  INSERT INTO loan (id, amount) VALUES (1, 100.00);
  ```

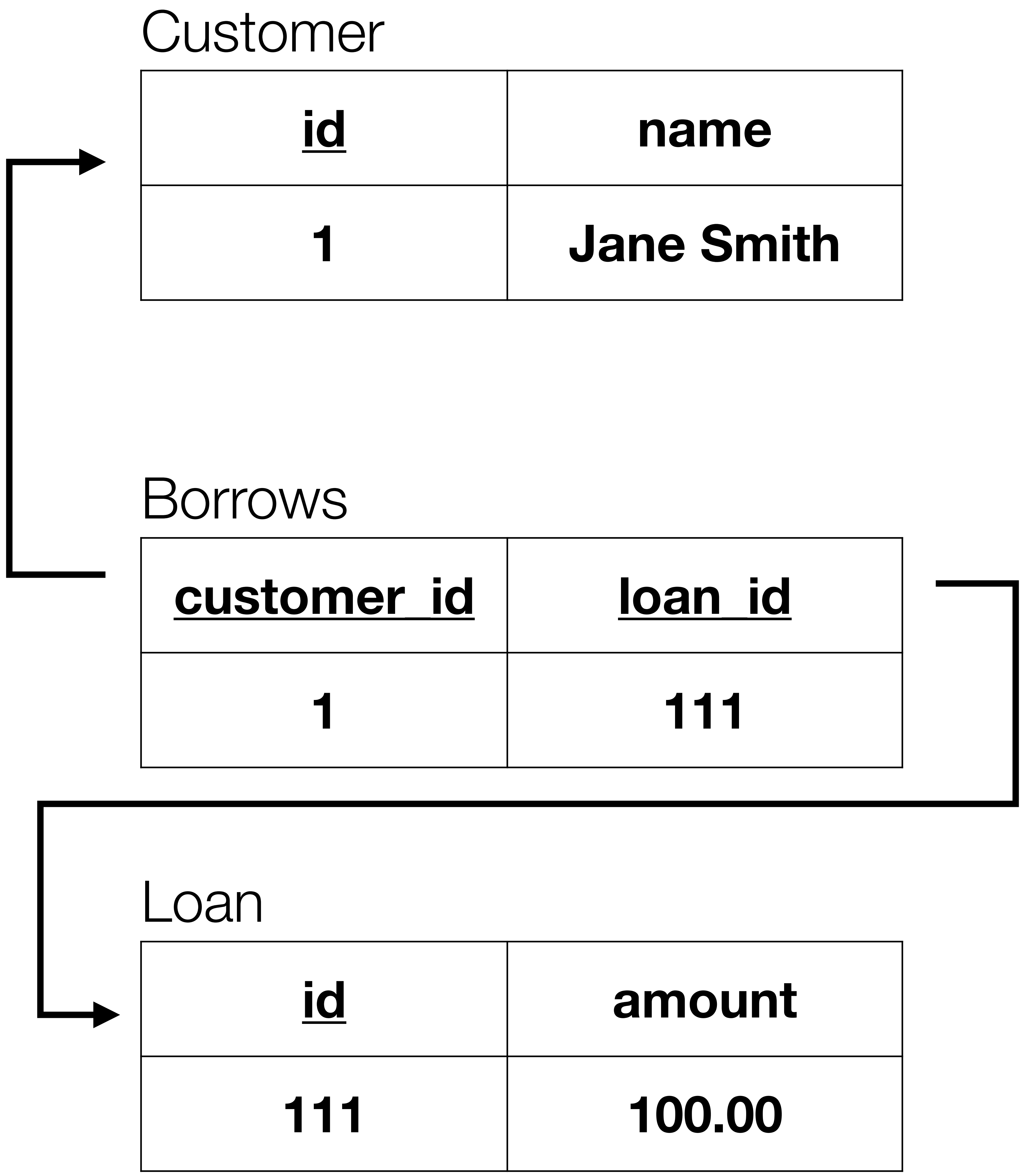
Today: Database Queries

- How to retrieve records from a database?
- Using the SQL DML (Structured Query Language Data Manipulation Language)
- Find the record for the loan with ID 111:

```
SELECT *
FROM loan
WHERE loan.id = 111;
```

- Supports sorting, queries across tables, computing averages, etc.
- Your SQL query tells the database what you want. The database (usually) retrieves the results as efficiently as possible.

Schema Review



Customer

id	name
1	Jane Smith

Borrows

customer_id	loan_id
1	111

Loan

id	amount
111	100.00

Queries with Relational Algebra

- What loans does Jane Smith have?
- For brevity, C = customer, B = borrows, L = loan
- $\pi_{\text{loan\_id, amount}}\left(\sigma_{\text{name=Jane Smith}}(C) \bowtie_{\text{C.id = B.customer\_id}} B \bowtie_{\text{B.loan\_id = L.id}} L\right)$
- $\pi_{\text{loan\_id, amount}}\left(\sigma_{\text{name=Jane Smith}}\left(C \bowtie_{\text{C.id = B.customer\_id}} B \bowtie_{\text{B.loan\_id = L.id}} L\right)\right)$

Query with Relational Calculus

-
  ```
  {l.id, l.amount | LOAN(l) AND
  ((∃b)(∃c)(BORROWS(b) AND CUSTOMER(c) AND
  l.id = b.loan_id AND b.customer_id = c.id AND
  c.name= 'Jane Smith'))}
  ```

Query with SQL

-
  ```
  SELECT loan_id, amount
  FROM customer
  JOIN borrows ON customer.id = borrows.customer_id
  JOIN loan ON borrows.loan_id = loan.id
  WHERE customer.name = "Jane Smith";
  ```

Why Three Query Languages?

- **Relational calculus:** declarative specification of a query
- **SQL:** user-friendly declarative specification of a query
- **Relational algebra:** imperative specification of a query
- To evaluate SQL, the DBMS chooses between multiple candidate relational algebra queries

Overview of DDL Operations

Operation	Statement
Create table	CREATE TABLE <name> (<field> <domain>, …)
Drop table	DROP TABLE <name>

Insert row into table	INSERT INTO <name> (<field names>)
	VALUES (<field values>)

Delete row from table	DELETE FROM <name>
	WHERE <condition>

Update rows in table	UPDATE <name>
	SET <field name> = <value>
	WHERE <condition>

Overview of DML Operations

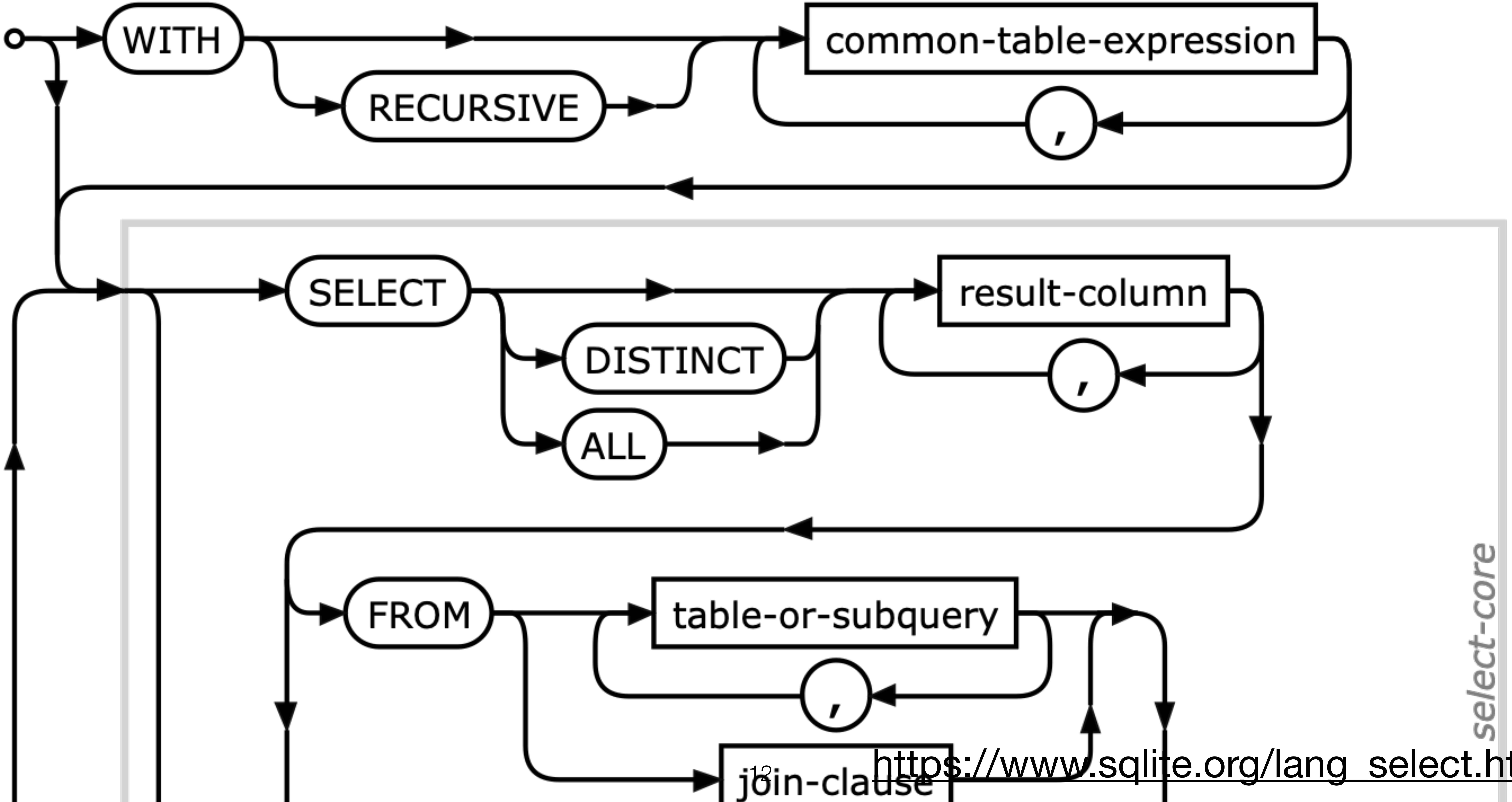
- Simple:

```
SELECT * FROM branch;
```

- Complex:

```
SELECT customer.id, SUM(amount) as debt
FROM customer
JOIN borrows ON customer.id = borrows.customer_id JOIN
loan ON borrows.loan_id = loan.id
GROUP BY customer.id
HAVING debt > 100
ORDER BY debt;
```

SELECT Documentation



https://www.sqlite.org/lang_select.html

SELECT FROM WHERE

-
  ```
  SELECT amount
  FROM loan
  WHERE amount > 1000
  ```

- How does this differ from:
  $\pi_{\text{amount}}\left(\sigma_{\text{amount}>1000}(\text{loan})\right)$

- Eliminating duplicates is costly, and sometimes duplicates are useful

loan

id	amount
111	100.00
112	9001.00
113	2000.00
114	2000.00

DISTINCT

- If you want to eliminate duplicates:

-
  ```
  SELECT DISTINCT amount
  FROM loan
  WHERE amount > 1000
  ```

- Relational algebra (RA) as we defined it works with sets. We could redefine it as a “bag algebra” to allow duplicates (RA*).

loan

id	amount
111	100.00
112	9001.00
113	2000.00
114	2000.00

SELECT FROM Multiple Tables

-
  ```
  SELECT name, loan_id
  FROM customer, borrows
  WHERE customer.id = borrows.customer_id
  ```

-
  ```
  SELECT name, loan_id
  FROM customer AS c, borrows AS b
  WHERE c.id = b.customer_id
  ```

- Similar to:

  $\pi_{\text{name, loan\_id}}\left(\sigma_{\text{customer.id = borrows.customer\_id}}(\text{customer} \times \text{borrows})\right)$

  $\pi_{\text{name, loan\_id}}\left(\text{customer} \bowtie_{\text{customer.id = borrows.customer\_id}} \text{borrows}\right)$

Conceptual Algorithm

-
  ```
  SELECT attribute1, attribute2, …
  FROM relation1, relation2, …
  [WHERE predicate]
  ```

- `FROM` could be implemented as a cartesian product, $\times$

- `WHERE` could be implemented as selection, σ

- `SELECT` could be implemented as projection, π

- $\pi_{a_1,a_2,\dots}\left(\sigma_P\left(r_1 \times r_2 \times \dots\right)\right)$

Advanced SELECT

- Use `*` to get all attributes
- Use `DISTINCT` to eliminate duplicates
- Use `AS` to rename columns
- Arithmetic operations are supported

-
  ```
  SELECT amount * 100 AS cents
  FROM loan
  WHERE amount > 1000
  ```

https://www.sqlite.org/lang_select.html

Advanced SELECT

- Use `ORDER BY` to sort results:

-
  ```
  SELECT amount
  FROM loan
  ORDER BY amount
  ```

- Aggregate operators are also available: `AVG`, `MIN`, `MAX`, `SUM`, `COUNT`, …

-
  ```
  SELECT SUM(amount)
  FROM loan
  ```

https://www.sqlite.org/lang_aggfunc.html

Advanced WHERE

- Predicates are composed of operators, attribute names, and constants

- Basic operators: <, >, <=, >=, =, !=, AND, OR, NOT, …

- SQL-specific operators: IN, LIKE, ISNULL, BETWEEN, …

-
  ```
  SELECT id
  FROM loan
  WHERE amount BETWEEN 9000 AND 10000 OR amount < 10
  ```

-
  ```
  SELECT name
  FROM customer
  WHERE name LIKE '% Smith'
  ```

https://www.sqlite.org/lang_expr.html

Advanced FROM

- Specifies which relation(s) to retrieve tuples from
- Can alias relations for convenience (and self-joins)
- Can directly specify joins

-
  ```
  SELECT name, loan_id
  FROM customer AS c JOIN borrows AS b
  ON c.id = b.customer_id
  ```

Preview: Query Evaluation Plans

-
  ```
  SELECT name, loan_id
  FROM customer, borrows
  WHERE customer.id = borrows.customer_id
  ```

- In three steps, draw on the board:

  $\pi_{\text{name, loan\_id}} \left(\sigma_{\text{customer.id = borrows.customer\_id}} (\text{customer} \times \text{borrows}) \right)$

customer

<u>id</u>	name
1	Jane Smith
2	John Smith
4	Hazel Jones

borrows

<u>customer_id</u>	<u>loan_id</u>
1	111
2	111
3	222

Preview: Query Evaluation Plans

-
  ```
  SELECT name, loan_id
  FROM customer, borrows
  WHERE customer.id = borrows.customer_id
  ```

- In two steps, draw on the board:

  $\pi_{\text{name, loan\_id}}\left(\text{customer} \bowtie_{\text{customer.id = borrows.customer\_id}} \text{borrows}\right)$

customer

id	name
1	Jane Smith
2	John Smith
4	Hazel Jones

borrows

customer_id	loan_id
1	111
2	111
3	222

Advanced SQL DML

CSCI 220: Database Management and Systems Design

Practice Quiz

- Use the SQL DML to form these queries:
  - Retrieve the names of all employees
  - Retrieve the name of the employee with SSN = 123456789
  - Retrieve the names and SSNs of the employees making more than $71,000
  - Retrieve the name of each manager and the name of the department they manage

employee

name	ssn	salary
John Smith	123	70000
Jane Smith	234	71000
Frank Wong	345	72000

department

name	id	mgr_ssn
Research	1	345
Administration	2	234

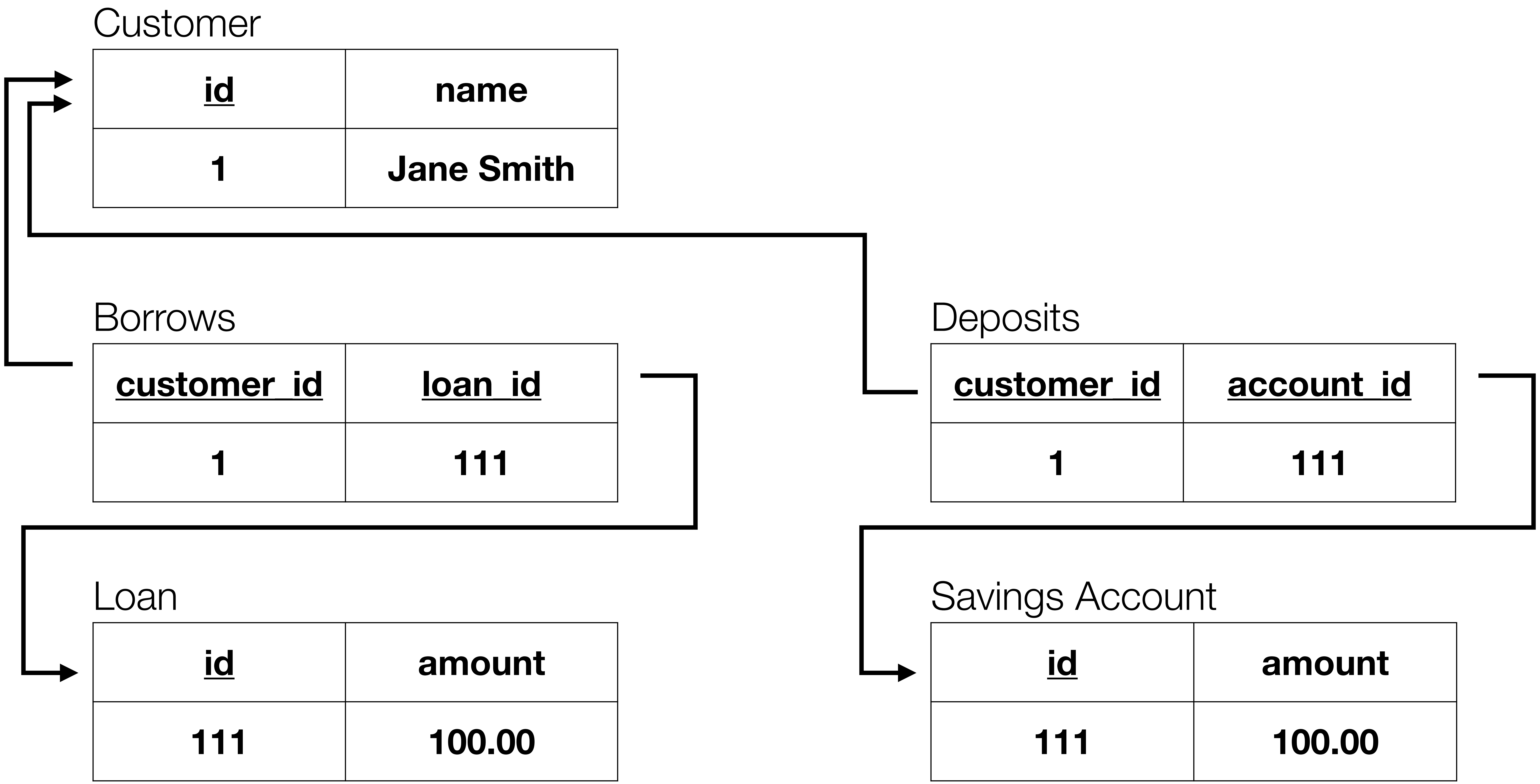
dept_locations

dept_id	Location
1	Houston
1	Boston
2	Boston

Today you will learn…

- How to use advanced SQL DML features

Schema Review



Customer

id	name
1	Jane Smith

Borrows

customer_id	loan_id
1	111

Deposits

customer_id	account_id
1	111

Loan

id	amount
111	100.00

Savings Account

id	amount
111	100.00

Aggregates and GROUP BY

- How to calculate the total debt of all customers?

```
SELECT customer.name, SUM(amount) as debt
FROM customer
JOIN borrows ON customer.id = borrows.customer_id
JOIN loan ON borrows.loan_id = loan.id
GROUP BY customer.id
```

HAVING

- How to calculate the total debt of all customers?
- For only those with more than $100 of debt?

```
SELECT customer.name, SUM(amount) as debt
FROM customer
JOIN borrows ON customer.id = borrows.customer_id
JOIN loan ON borrows.loan_id = loan.id
GROUP BY customer.id
HAVING debt > 100
```

Views

- How to calculate the total debt of all customers?

- Save the query as a view:

```
CREATE VIEW debt_view AS
SELECT customer.id, customer.name, SUM(amount) as debt
FROM customer
JOIN borrows ON customer.id = borrows.customer_id
JOIN loan ON borrows.loan_id = loan.id
GROUP BY customer.id;

SELECT * FROM debt_view;
```

INTO

- Copy data into a new table:
```
SELECT *
INTO customer_2024-1-1_bak
FROM customer
```

- Not supported by SQLite, but equivalent to:
```
CREATE TABLE customer_2024-1-1_bak AS
SELECT *
FROM customer
```

SERIAL

- How to assign unique identifiers to records?
  - For example: customer.id, loan.id, etc.
- In PostgreSQL:

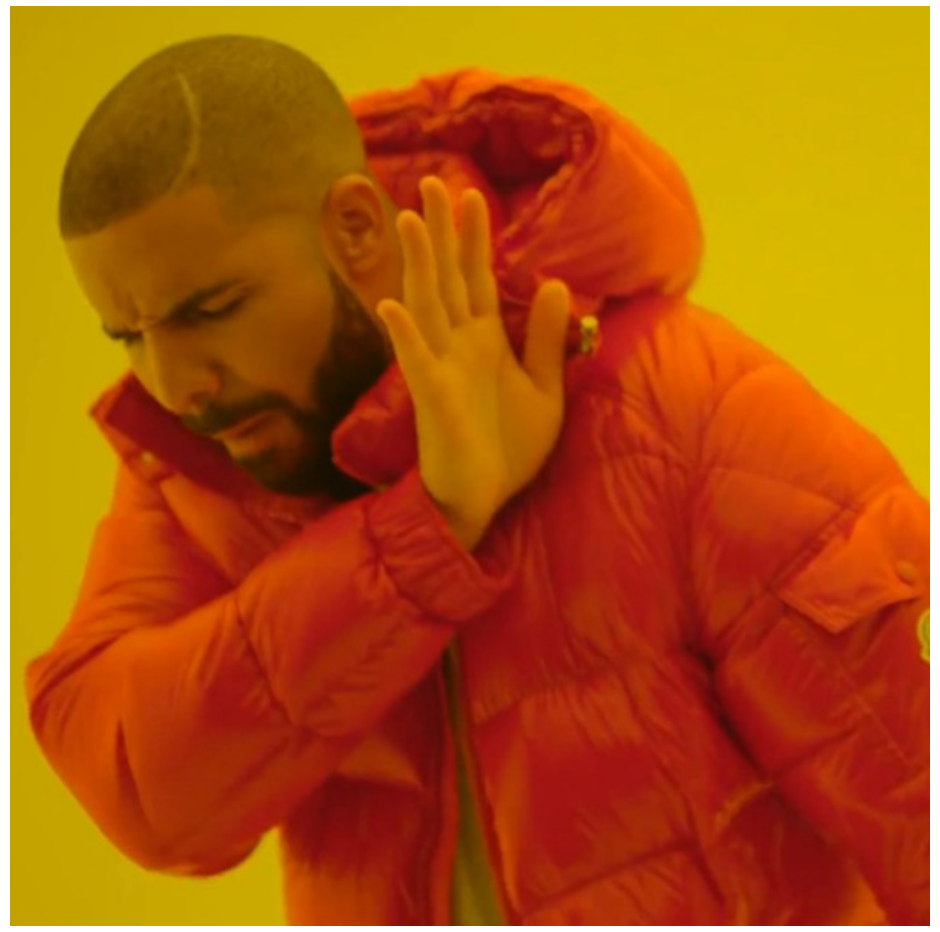
```
CREATE TABLE customer (id SERIAL PRIMARY KEY, name TEXT);

INSERT INTO customer (id, name) VALUES (DEFAULT, 'Jane Smith');
INSERT INTO customer (name) VALUES ('John Smith');
```

- SQLite uses AUTOINCREMENT and ROWID to similar effect

Review: NULL

- Databases offer a special value, NULL
- NULL can be used to represent unknown or inapplicable values
- For example, a newly hired employee’s HIRE_DATE may be NULL until it is decided
- Only allow NULL if you need to
    - By default, all columns can contain NULL (except primary key columns)

Locating NULL Values



Doesn’t work:

```
SELECT * FROM employee
WHERE hired = NULL
```

Instead:

```
SELECT * FROM employee
WHERE hired IS NULL
```

employee

id	name	hired
1	Peter	2023-1-1
2	Sara	2023-5-1
3	Drake	NULL

NULL Operations

Expression	Result
1 + NULL	NULL
1 - NULL	NULL
1 * NULL	NULL
1 / NULL	NULL
1 = NULL	NULL
1 < NULL	NULL
1 > NULL	NULL
TRUE OR NULL	**TRUE**
TRUE AND NULL	NULL
FALSE OR NULL	NULL
FALSE AND NULL	**FALSE**
NULL AND NULL	NULL
NULL OR NULL	NULL

Some DBMSs use a third boolean state, UNKNOWN, instead of NULL. IMHO, NULL is clearer.

Sets vs Multisets

- Relational algebra (RA) operates on sets
- SQL DML operates on **multisets** (AKA, bags)
  - Duplicates are preserved (by default)
  - Relational algebra can be extended to work on multisets (RA*)

RA* Examples

- Additive addition: $R \cup^* S$
- Bag difference:
  $R -^* S$
  $S -^* R$
- Also think about:
  $\sigma^*, \pi^*, \times^*$

R

A	B
1	y
1	y
2	z

S

A	B
1	y
2	z
3	y

SQL Set and Bag Operations

Set Operation	Bag Operation
UNION	UNION ALL
INTERSECT	INTERSECT ALL
EXCEPT or MINUS	EXCEPT ALL or MINUS ALL

```
(SELECT customer_id FROM borrows)
      UNION/INTERSECT/MINUS
(SELECT customer_id FROM deposits)
```

Nested Queries

- An (outer) query can contain (inner) queries in the **FROM** or WHERE clause

- Find loans, except those with id 222:
```
SELECT id
FROM loan
EXCEPT (SELECT id FROM loan WHERE loan.id = 222)
```

How can this be written without a subquery?

- Find the ID of the customer with the most debt:
```
SELECT customer_id, MAX(debt)
FROM
(SELECT customer_id, SUM(amount) as debt
FROM borrows JOIN loan ON borrows.loan_id = loan.id
GROUP BY customer_id)
```

Nested Queries

- An (outer) query can contain (inner) queries in the FROM or **WHERE** clause

- Find the largest loans:
```
SELECT id, amount
FROM loan
WHERE loan.amount = (SELECT MAX(amount) FROM loan)
```

- (Mostly) equivalent to:
```
SELECT id, amount
FROM loan
WHERE loan.amount >= ALL (SELECT amount FROM loan)
```

Nested Queries

- Common operators:
  UNION, INTERSECT, EXCEPT, IN, ALL, ANY, EXISTS, UNIQUE

- Bag operators (keep duplicates):
  UNION ALL, INTERSECT ALL, EXCEPT ALL

JOINs

- We use joins to associate records across relations
- Inner joins: records without associated records **are omitted**
- Outer joins: records without associated records **are retained**
    - LEFT ⟕, RIGHT ⟖, and FULL ⟗

Inner Join

Customer

id	name
1	Sam Wilson
2	Steve Rogers
3	Peggy Carter

Borrows

customer_id	loan_id
1	111
2	222

```
SELECT *
FROM customer
JOIN borrows ON customer.id = borrows.customer_id
```

id	name	customer_id	loan_id
1	Sam Wilson	1	111
2	Steve Rogers	2	222

Left Outer Join

Customer

<u>id</u>	name
1	Sam Wilson
2	Steve Rogers
3	Peggy Carter

Borrows

<u>customer_id</u>	<u>loan_id</u>
1	111
2	222

```
SELECT *
FROM customer
LEFT OUTER JOIN borrows ON customer.id = borrows.customer_id
```

id	name	customer_id	loan_id
1	Sam Wilson	1	111
2	Steve Rogers	2	222
3	Peggy Carter	NULL	NULL

Left Outer Join

Customer

id	name
1	Sam Wilson
2	Steve Rogers
3	Peggy Carter

Borrows

customer_id	loan_id
1	111
2	222

```
SELECT *
FROM borrows
LEFT OUTER JOIN customer ON customer.id = borrows.customer_id
```

id	name	customer_id	loan_id
1	Sam Wilson	1	111
2	Steve Rogers	2	222

Full Outer Join

Customer

id	name
1	Sam Wilson
3	Peggy Carter

(Deleted Steve Rogers)

Borrows

customer_id	loan_id
1	111
2	222

```
SELECT *
FROM customer
FULL OUTER JOIN borrows ON customer.id = borrows.customer_id
```

id	name	customer_id	loan_id
1	Sam Wilson	1	111
3	Peggy Carter	NULL	NULL
NULL	NULL	2	222

Cross Product

Customer

id	name
1	Sam Wilson
2	Steve Rogers
3	Peggy Carter

Borrows

customer_id	loan_id
1	111
2	222

```
SELECT *
FROM customer, borrows
```

id	name	customer_id	loan_id
1	Sam Wilson	1	111
1	Sam Wilson	2	222
2	Steve Rogers	1	111
2	Steve Rogers	2	222
3	Peggy Carter	1	111
3	Peggy Carter	2	222

Review: Kitchen Sink Query

```
SELECT customer.id, SUM(amount) as debt
FROM customer
JOIN borrows ON customer.id = borrows.customer_id
JOIN loan ON borrows.loan_id = loan.id
GROUP BY customer.id
HAVING debt > 100
ORDER BY debt
```

SQL DML vs RA

Clause	Evaluation Order	Relational Algebra
SELECT [DISTINCT]	4	$\pi^{[*]}$
FROM	1	X^*
WHERE	2	σ^*
INTO	7	$\leftarrow$
GROUP BY	3	$\mathfrak{I}^*$
HAVING	5	$\sigma^*(\rho^*(\mathfrak{I}^*(\ldots)))$
ORDER BY	6	Can’t express