Relation Schema

- Given domains D_1, D_2, …. D_n a **relation** *r* is a subset of

  D_1 x D_2 x … x D_n (*cartesian product*)

  Thus, a relation is a set of tuples (a1, a2, …, an)

  where each ai $\in$ Di

- Schema of a relation consists of
  - attribute definitions
    - name
    - type/domain
  - integrity constraints

Tuple ⟶ **Row**

Relation ⟶ **Table**

Math General

Schema and Relations

`Account_schema = (account_number, branch_name, balance)`

Schema

`account(Account_schema)`

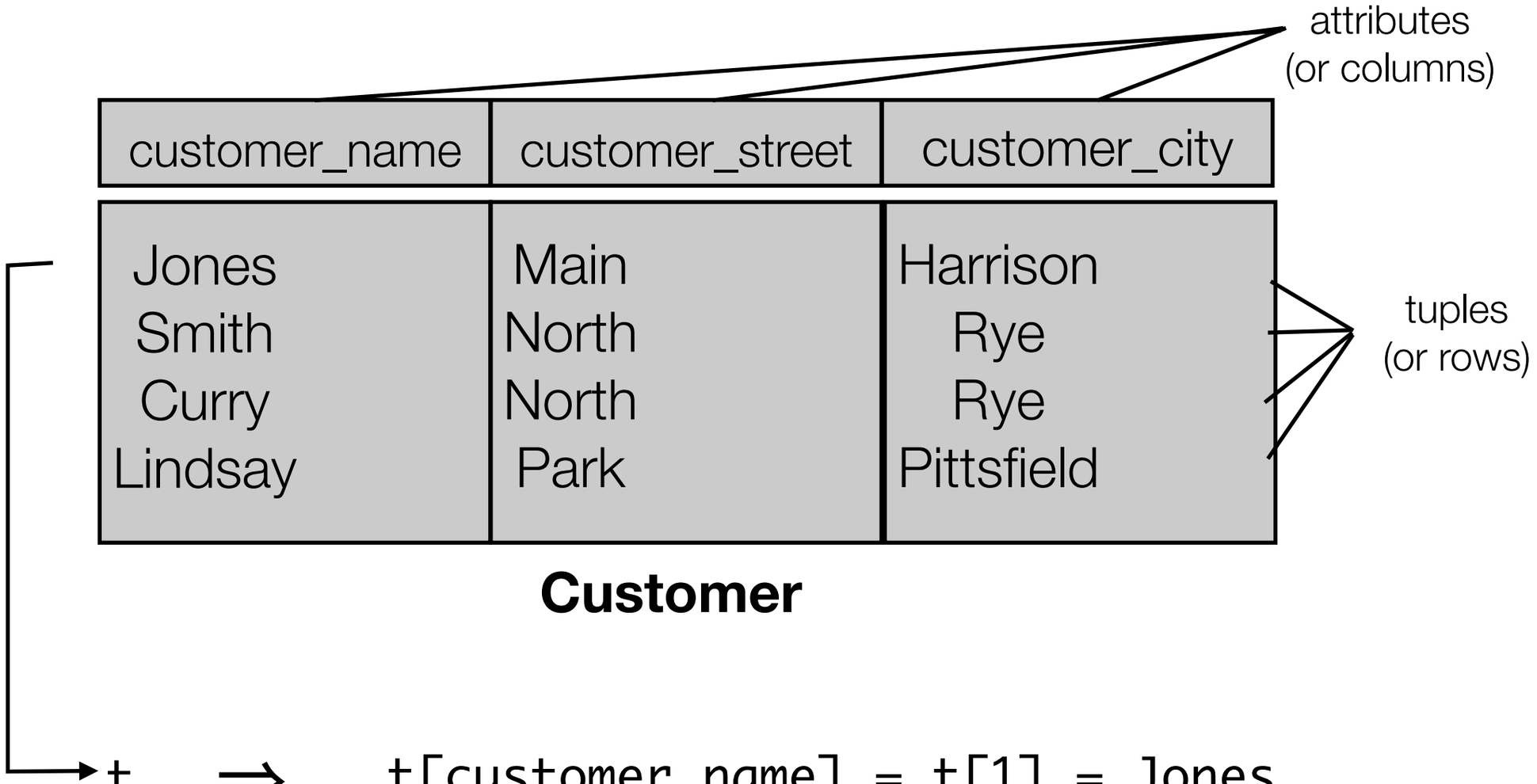
Relation from a schema

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Relation **instance**

Relation Instance

- The current values (relation instance) of a relation are specified by a table
- An element t of r is a tuple, represented by a row in a table
- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)



Customer

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information
- E.g.

  `account` : information about accounts

  `depositor` : which customer owns which account

  `customer` : information about customers

The `customer` Relation

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

```
Customer_schema = (customer_name, customer_street, customer_city)
```

The `depositor` Relation

customer_name	*account_number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

```
Depositor_schema = (customer_name, account_number)
```

Why Split Information Across Relations?

- Storing all information as a single relation such as

  ```
  Bank_schema = (account_number, balance, customer_name, ..)
  ```

- Results in
  - repetition of information
    - e.g.,if two customers own an account (What gets repeated?)
  - the need for null values
    - e.g., to represent a customer without an account
- Normalization theory (Chapter 7) deals with how to design relational schemas

Keys

- Reflect constraints in the real-world enterprise
- Let K ⊆ R
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation r(R)
  - by “possible r ” we mean a relation r that could exist in the enterprise we are modeling.
  - Example: {customer_name, customer_street} and
    {customer_name}
    are both superkeys of Customer, if no two customers can possibly have the same name
    - In real life, an attribute such as customer_id would be used instead of customer_name to uniquely identify customers, but we omit it to keep our examples small, and instead assume customer names are unique.

Keys (Cont.)

- K is a **candidate key** if K is *minimal*: no subset of K is a superkey
  Example: {customer_name} is a candidate key for Customer
- Primary key: a candidate key chosen as the principal means of identifying tuples within a relation
  - Should choose an attribute whose value never, or very rarely, changes.
  - E.g. email address is unique, but may change
  - Others?
  - Generate your own

Keys and schema

R: relational schema

K: superkey of **R** ⇒ restriction on relations **r(R)**

$t_1, t_2 \in r$ and $t_1 \neq t_2 \Rightarrow t_1[K] \neq t_2[K]$

Foreign Keys

- A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a **foreign key**.
  - E.g. `customer_name` and `account_number` attributes of `depositor` are foreign keys to `customer` and `account` respectively.
  - Only values occurring in the primary key attribute of the referenced relation may occur in the foreign key attribute of the referencing relation.

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye

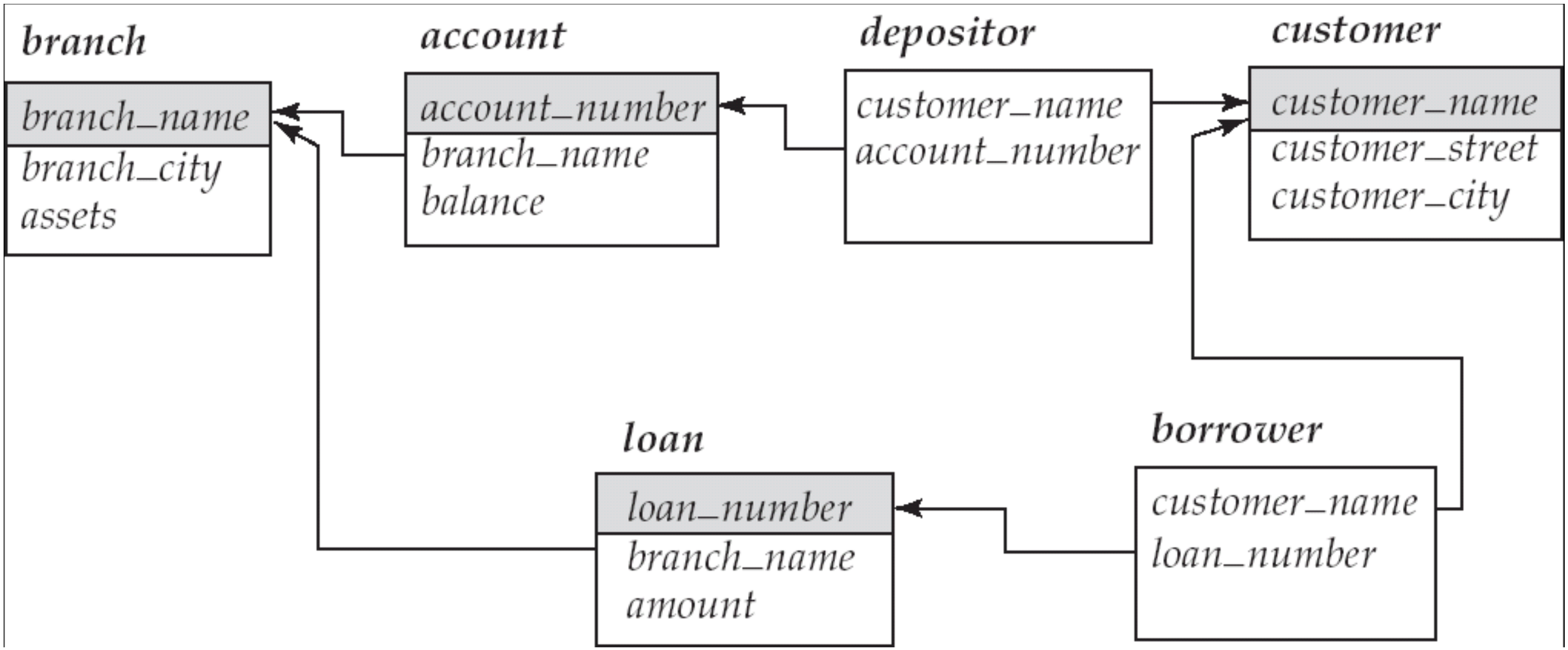
account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900

customer_name	*account_number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Referenced relation

Referencing relation

Schema Diagram



Primary key

Query Languages

- Language in which user requests information from the database.
- Categories of languages
    - **Procedural**
    - Non-procedural, or **declarative**
- “Pure” languages:
    - Relational algebra
    - Tuple relational calculus
    - Domain relational calculus
- Pure languages form underlying basis of query languages that people use.

Relational Algebra

- Similar to regular algebra (3*x + 2*y)
- Relations instead of numbers
- Why study?
  - foundation of low-level DBMS operations
  - in order to understand query execution
  - Build sophisticated SQL queries
- More procedural than SQL (which is declarative)

Set Theory

- A set: **unordered** collection of **distinct** objects
- Elements of a set
- Subset, proper subset, superset
- Union
- Intersection
- set difference
- Cartesian product (set of ordered pairs)

```
{0, 20, 12, 60}
{Canada, U.S.A.}
```

Relational Operators

- Six basic operators
  - select: σ
  - project: ∏
  - union: ∪
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Select Operation – Example

All tuples in relation `loan` where branch is "Perryridge"

$$\sigma_{\texttt{branch\_name="Perryridge"}}(\texttt{loan})$$

Predicate

loan_number	branch_name	amount
L-15	Perryridge	1500
L-16	Perryridge	1300

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

All tuples with amount lent is more than $1200

$$\sigma_{\texttt{amount > 1200}}(\texttt{loan})$$

Select Operation

Comparators: =, ≠, <, ≤, >, ≥

Connectives: and (∧), or (v), not (¬)

$$\sigma_{\text{branch\_name}=\text{“Perryridge”} \wedge \text{amount} > 1350}(\text{loan})$$

loan_number	branch_name	amount
L-15	Perryridge	1500

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Project Operation – Example

List only part of a relation

$$\pi_{\texttt{loan\_number, amount}}(\texttt{loan})$$

loan_number	amount
L-11	900
L-14	1500
…	…

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Composition of operations

Result of a relational operation is a relation

$$\pi_{\texttt{loan\_number}}(\sigma_{\texttt{branch\_name="Perryridge"}}(\texttt{loan}))$$

loan_number
L-15
L-16

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Union operation

- Combine the result of two operations

Query: customers with an account or a loan (or both)

Depositor relation

customer_name	account_number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Borrower relation

customer_name	account_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

$$\Pi_{customer\_name}(depositor) \cup \Pi_{customer\_name}(borrower)$$

customer_name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Rules for Union compatibility

- For `(r ∪ s)` to work
  - `r` and `s` should have **same arity** (same number of attributes)
  - corresponding attributes should have **same domain**

Set-difference operations

- Find tuples in one that are not present in another

$$\Pi_{customer\_name}(depositor) - \Pi_{customer\_name}(borrower)$$

customer_name
Lindsay
Johnson
Turner

- Same rules for compatibility apply as in Union

Cartesian-Product operation

- Combine information, r_1 x r_2
- Remember: a relation is a subset of a Cartesian product
- Naming scheme to differentiate attributes: `relation.attribute`
    - only for non-distinct attributes
- What tuples appear in r_1 x r_2 ?
- tuples in r_1 x r_2 : all possible combinations of tuples in r_1 and r_2
- if r_1 has n_1, and r_2 has n_2, then r_1 x r_2 has $n_1 * n_2$ tuples

Cartesian-Product

- For relations $r_1(R_1)$, $r_2(R_2)$:
  - r_1 x r_1 concatenation of R_1 and R_2
- For tuple $t \in r_1$ x r_1,, then:
  - $t[R_1] = t_1[R_1]$ and $t[R_2] = t_2[R_2]$