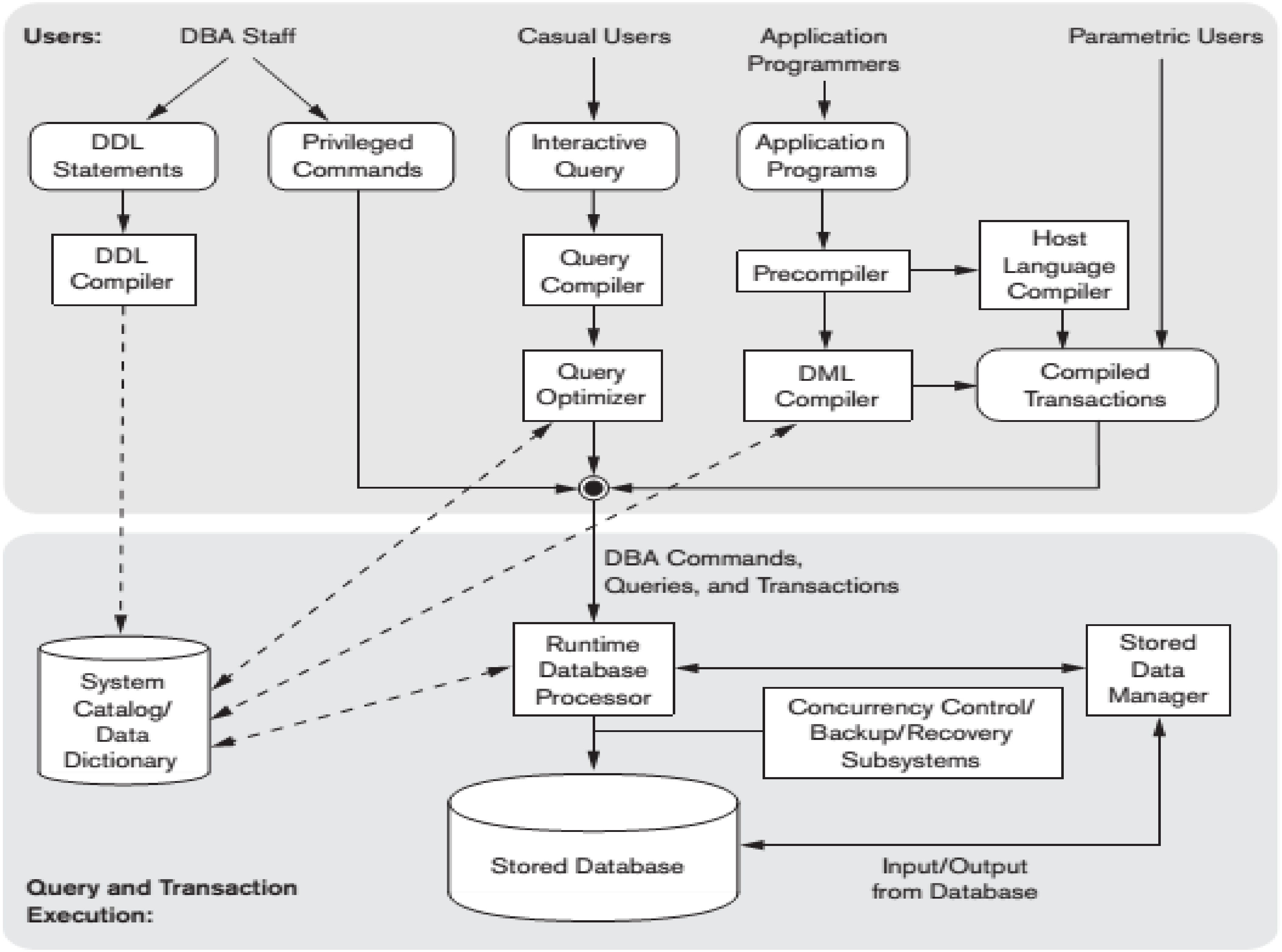


Component Models of a DBMS and their interfaces

Different Actors of Database System

Database Administrators:

- **Schema definition: Executing Data Definition Language(DDL)**
- **Storage Structures and access method definition**
- **Schema and physical-organization modification**
- **Granting of authorization for data access**
- **Routine maintenance:**
  - **Periodically backing the database to prevent from loss of data**
  - **Monitoring jobs running on the database and ensuring performance**
  - **Ensuring free space availability for normal operations**

Different Actors of Database System

Database Designers:

- Database are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.

- These tasks are mostly undertaken before the database is actually implemented and populated with data.

- It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements

Database Systems...contd.

End User: End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.

- **Casual end users:** Occasionally access the database, but they may need different information each time.

  - They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.

- **Naïve or parametric end users:**

  - Unsophisticated user who interact with the system by invoking one of the application that have been written previously. Make up a sizable portion of database end users.

  - Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called canned transactions—that have been carefully programmed and tested.

  - The tasks that such users perform are varied:
    - Bank tellers check account balances and post withdrawals and deposits.
    - Reservation agents for airlines, hotels, and car rental companies check a availability for a given request and make reservations

Database Systems...contd.

Sophisticated end users: Include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

Standalone users: Maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

- **An example is the user of a tax package that stores a variety of personal financial data for tax purposes.**

- **System Analysts and Application Programmer:**
  - **System analysts determine the requirements of end users, especially naive and parametric end users.**
  - **They develop specifications for standard canned transactions that meet these requirements.**
  - **Canned Transaction is the process of constantly querying and updating the database using standard types of queries and updates**
  - **Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.**

Data Manipulation Language:

- Language for accessing and manipulating the data organized by the appropriate data model. DML also known as query language

-The types of access are:

- Retrieval of information stored in the database
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

Two classes of languages

- Procedural DML– User specifies what data is required and how to get those data
- Declarative (nonprocedural)DML–User specifies what data is required without specifying how to get those data

Database Systems...contd.

- **General DML statements are:**

SELECT - retrieve data from the a database

INSERT - insert data into a table

UPDATE - updates existing data within a table

DELETE - deletes all records from a table, the space for the records remain

MERGE - UPSERT operation (insert or update)

CALL - call a PL/SQL or Java subprogram

EXPLAIN PLAN - explain access path to data

LOCK TABLE - control concurrency

Database Systems...contd.

- **Data Definition Language**
- **Specification notation for defining the database schema.**
- **DDL also used to specify:**
  - **Domain constraints: Domain of possible value must be associated with every attribute**
  - **Referential integrity: The value that appears in one relation for a given set of attributes should also appears in the another relation for a certain set of attributes**
  - **Assertion: Assertion is any condition that the database must always satisfy.**
  - **Authorization: Read authorization, write authorization, update authorization, delete authorization.**
  - **Example:**

```
create table account (
                account_number   char(10),
                branch_name      char(10),
                balance          integer)
```

- **Referential integrity** – **A foreign key must have a matching primary key or it must be null**

-This constraint is specified between two tables (parent and child); it maintains the correspondence between rows in these tables. It means the reference from a row in one table to other table must be valid.

- **Examples**

  - In the Customer/Order database: Customer(**<u>custid</u>**, custname)
    Order(**<u>orderID</u>**, custid, OrderDate)
  - To ensure that there are no orphan records, we need to enforce referential integrity.
  - An orphan record is one whose foreign key value is not found in the corresponding entity – the entity where the PK is located.
  - Recall that a typical join is between a PK and FK.

  - **The referential integrity constraint states that the CustID in the Order table must match a valid CustiD in the Customer table. Most relational databases have declarative referential integrity.**

- When the tables are created the referential integrity constraints are set up.

Database Management System Internals

- Storage management
- Query processing
- Transaction processing

Storage Management

- **The main components of storage manager are:**
  - **Authorization and Integrity Manager: Checks integrity constraints & authority of user**
  - **Transaction Manager: Ensures database remains in constant state despite system failure**
  - **File Manager: Manages allocation of free space**
  - **Buffer Manager: Responsible for fetching data from storage into main memory**

- **Storage manager implements several data structures as a part of physical implementation**
  - **Data Files: Stores database itself**
  - **Data Dictionary: Stores metadata**
  - **Indices: Which provides fast access to data items**

Query Processor: Query processing components includes:

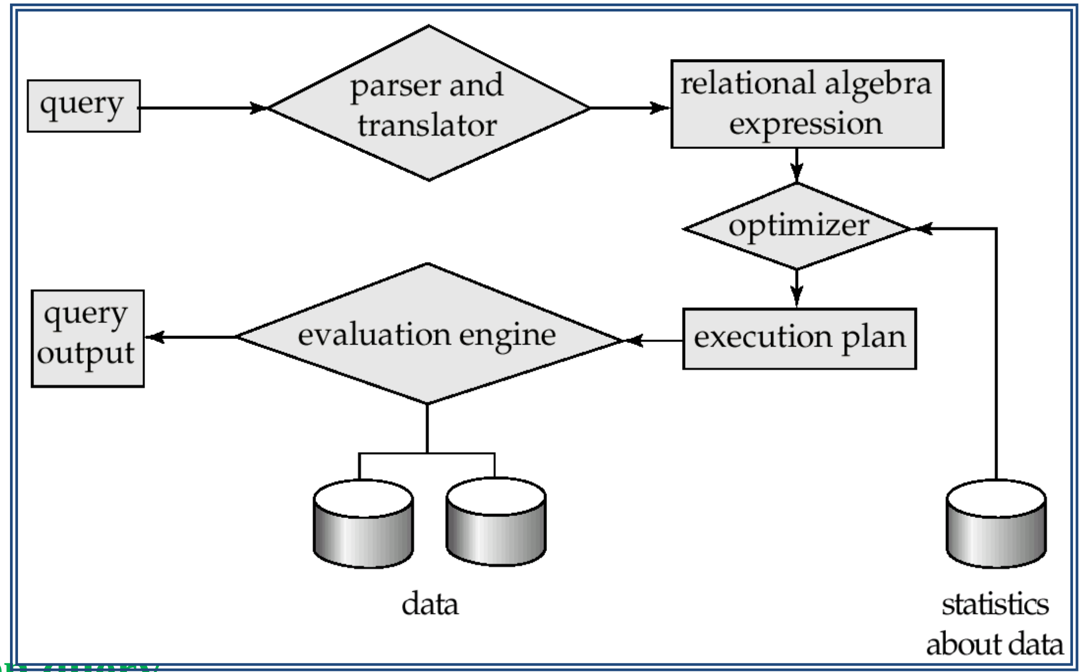
- **DDL Interpreter:** Which interprets DDL statements and records the definition in the data dictionary

- **DML Compiler:** Which translates DML statements in a query language into a evaluation plan consisting of low-level instructions that query evaluation engine understand.

  -DML compiler can also perform query optimization

- **Query Evaluation Engine:** Which executes low-level instructions generated by the DML compiler

Query processing



It includes following steps:

1. **Storage Parsing and Translation**

2. **Optimization**

3. **Evaluation**

- **Alternative ways of evaluating a given query**
  - **- Equivalent expressions**
  - **- Different algorithms for each operation**
- **Cost difference between a good and a bad way of evaluating a query can be enormous**
- **Need to estimate the cost of operations**
  - **- Depends critically on statistical information about relations which the database must maintain**
  - **- Need to estimate statistics for intermediate results to compute cost of complex expressions**

Transaction Management

- A transaction is a collection of operations that performs a single logical function in a database application

- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures: power failures and operating system crashes) and transaction failures.

- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

DBMS Interfaces

- **Menu-Based Interfaces for web Clients or Browsing**
- **Forms-Based Interfaces for Naïve users**
- **Graphical User Interface**
- **Natural Language Interface**
- **Interface for DBA**
- **Interface for Parametric User**

Database Systems...contd.

- **Main inhibitors (costs) of using a DBMS:**
  - **High initial investment and possible need for additional hardware.**
  - **Overhead for providing generality, security, concurrency control, recovery, and integrity functions.**

- **When a DBMS may be unnecessary:**
  - **If the database and applications are simple, well defined, and not expected to change.**
  - **If there are stringent real-time requirements that may not be met because of DBMS overhead.**
  - **If access to data by multiple users is not required.**

- **When no DBMS may suffice:**
  - **If the database system is not able to handle the complexity of data because of modeling limitations**
  - **If the database users need special operations not supported by the DBMS.**

Relational Model

- The relational model uses a collection of tables to represent both data & the relationships among those data.

- Each table has multiple columns, and column has a unique name. Each attribute of a relation has a name.

- The set of allowed values for each attribute is called the domain of the attribute.

- Attribute values are required to be atomic; that is, indivisible

  - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers

- Domain is said to be atomic if all its members are atomic

- The special value null is a member of every domain. The null value causes complications in the definition of many operations

Relational Model...contd.

Database schema: **The database schema is a logical design of 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

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

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

Account Relation

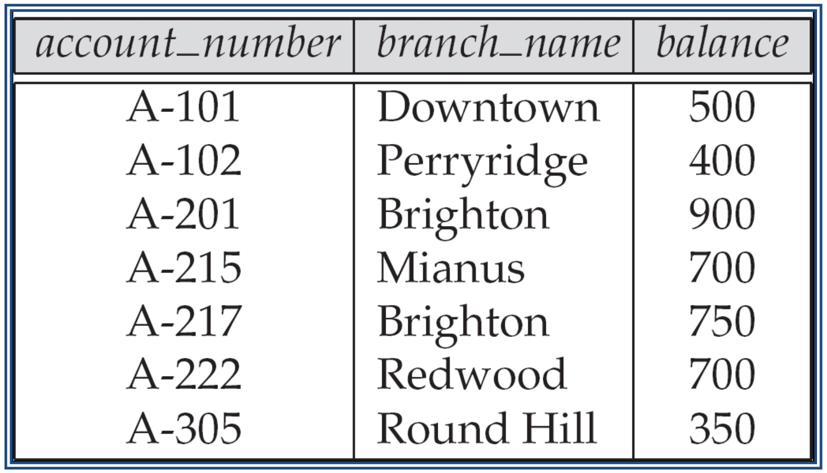
account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Relational Model…contd.

Because tables are relations, we use mathematical terms relation and tuple in place of table and row

Comparison of database terms with programming language terms

- **The concept of relation corresponds to the programming language notion of the variable.**
- **The concept of relation schema corresponds to the programming language notion of the type definition.**
- **The concept of relation instance corresponds to the programming language notion of a value of the variable.**



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

Relational Model...contd.

Relation Schema

- Formally, given domains $D_1, D_2, \dots. D_n$ a relation r is a subset of $D_1 \times D_2 \times \dots \times D_n$
  Thus, a relation is a set of n-tuples $(a_1, a_2, \dots, a_n)$ where each $a_i \in D_i$
- A relation is a subset of Cartesian product of a list of domains
- Schema of a relation consists of
  - attribute definitions
    - name
    - type/domain
  - integrity constraints

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

attributes
(or columns)

customer_name	*customer_street*	customer_city
Jones	Main	Harrison
Smith	North	Rye
Curry	North	Rye
Lindsay	Park	Pittsfield

tuples
(or rows)

customer

Relational Model…contd.

Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the domain of the attribute
- Attribute values are normally required to be atomic; that is, indivisible
  - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are atomic
- The special value null is a member of every domain
- The null value causes complications in the definition of many operations

Relational Model...contd.

Why Split Information Across Relations

Storing all information as a single relation such as

bank(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 deals with how to design relational schemas

Relational Model...contd.

Keys

- Let $K \subseteq R$ ***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.**

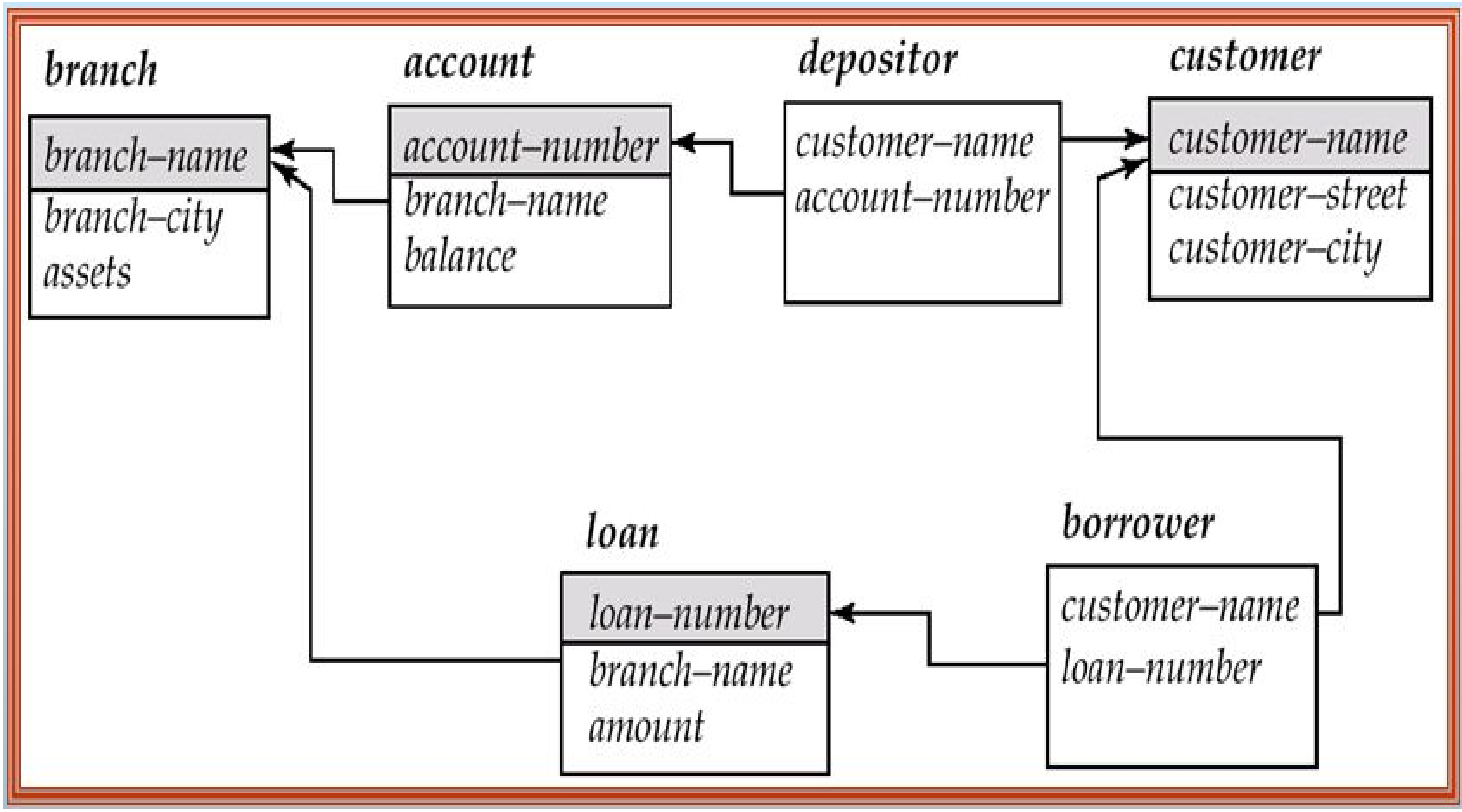
- **Example:{*customer_name,customer_street*} and**
  {*customer_name*}

  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.**

- **A superkey is a set of one or more attributes that, taken collectively identify uniquely a tuple in the relation.**

Relational Model…contd.



Banking Enterprise

Relational Model…contd.

- K is a candidate key if K is minimal

Example: {customer_name} is a candidate key for Customer, since it is a superkey and no subset of it is a superkey.

- **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**

- **Foreign Key: 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.

Relational Model…contd.

- **Query Languages**
- **A query language is a language in which a user requests information from the database.**
- **In procedural language the user interacts the system to perform a sequence of operations on the database to perform result.**
- **The relational algebra is a pure procedural language.**
- **Relational algebra gives formal foundation for relational model operations**
- **It is used as basis for implementing and optimizing queries in the query processing & optimization model**

Relational Model...contd.

- **Some of the relational algebra concepts are incorporated into the SQL standard query language for RDBMS.**

- **The basic set of operations for relational model is the relational algebra.**

- **A sequence of relation algebra operations forms a relational algebra expression**

- **The result of a retrieval is a new relation which may have been formed from one or more relations**

- **The fundamental operations in the relational algebra are Select, Project, Union, Setdifference, Cartesian product and Rename.**

Relational Model...contd.

Procedural language

Six basic operators

- select: σ
- project: $\prod$
- union: $\cup$
- set difference: –
- Cartesian product: x
- rename: ρ

The operators take one or two relations as inputs and produce a new relation as a result.

Select Operation

- Relation r

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

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

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Union Operation

- **Relations *r*, *s*:**

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- **r $\cup$ s:**

A	B
α	1
α	2
β	1
β	3

Set Difference Operation

- **Relations *r*, *s*:**

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- ***r* – *s*:**

A	B
α	1
β	1

Cartesian-Product Operation

- Relations *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

- *r* x *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*

Rename Operation

- **Allows us to name, and therefore to refer to, the results of relational-algebra expressions.**
- **Allows us to refer to a relation by more than one name.**
- **Example:**

$$\rho_x(E)$$

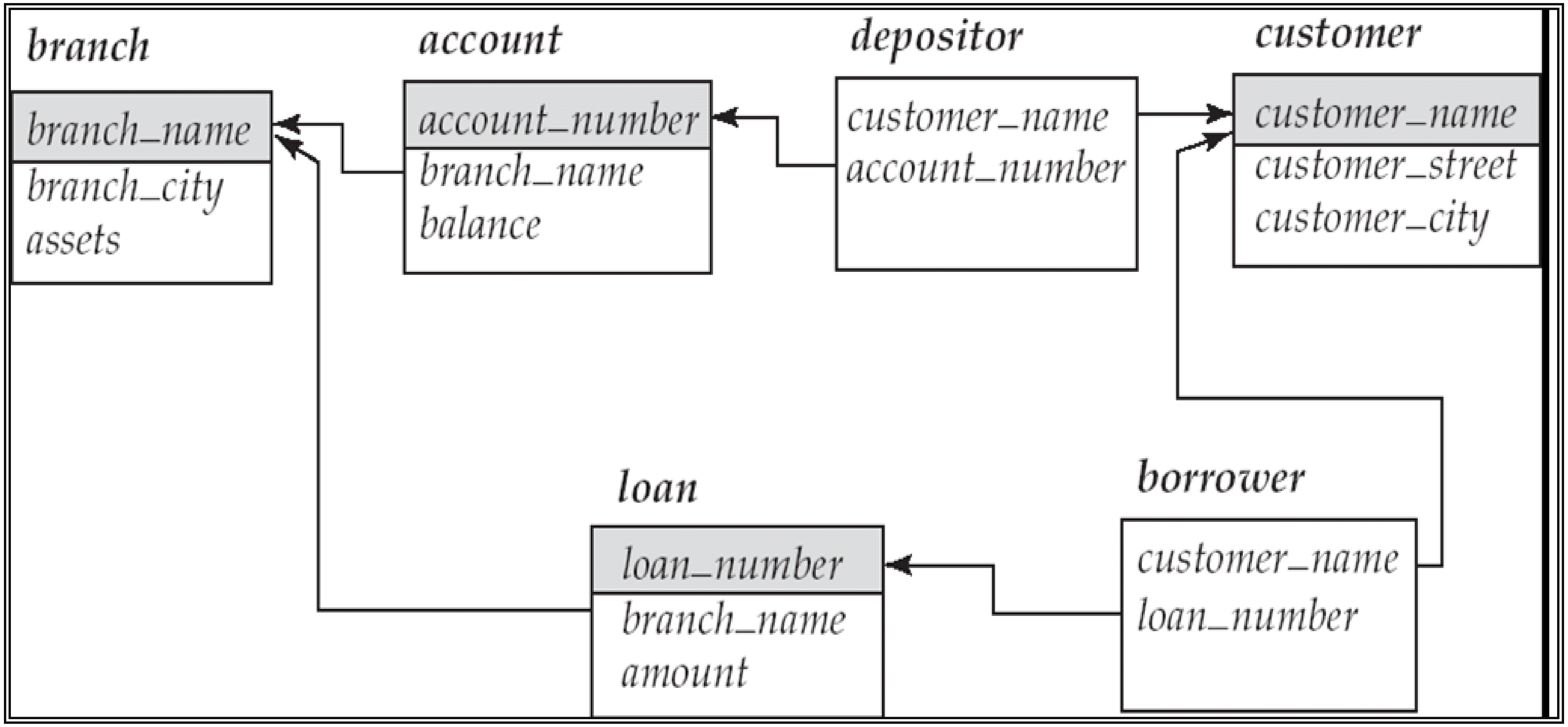
returns the expression *E* under the name *X*

- **If a relational-algebra expression *E* has arity *n*, then**

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

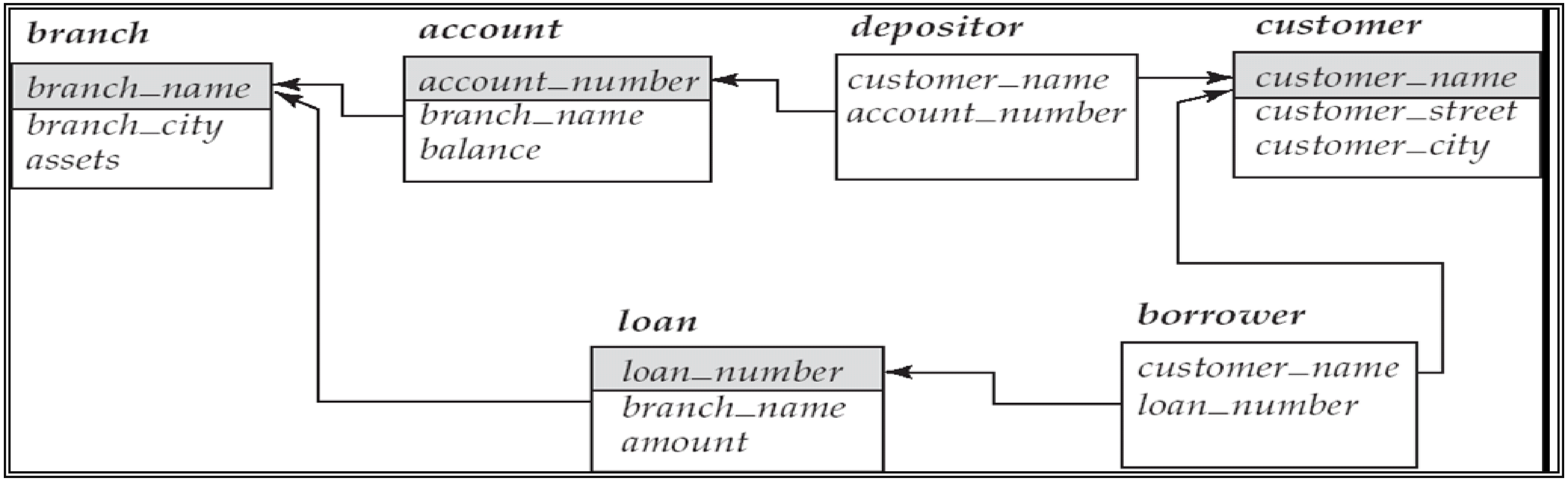
returns the result of expression *E* under the name *X*, and with the attributes renamed to A_1, A_2, …., A_n .

Relational Algebra Queries:



- **Find all loans of over Rs 1200**
- **Find the loan number for each loan of an amount greater than Rs1200**
- **Find the names of all customers who have a loan, an account, or both, from the bank**

Relational Algebra Queries:



- **Find all loans of over Rs1200**

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

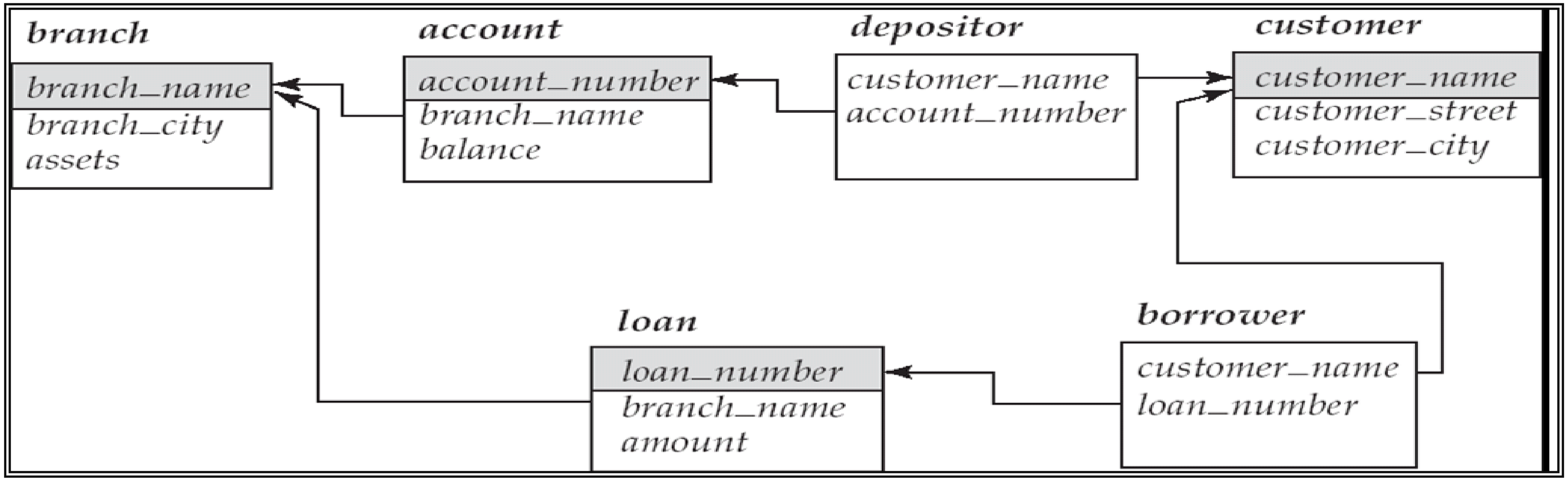
- **Find the loan number for each loan of an amount greater than Rs1200**

$$\Pi_{loan\_number}(\sigma_{amount > 1200}(loan))$$

- **Find the names of all customers who have a loan, an account, or both, from the bank**

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

Project Operation



- **Find the names of all customers who have a loan at the Perryridge branch.**
- **Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.**

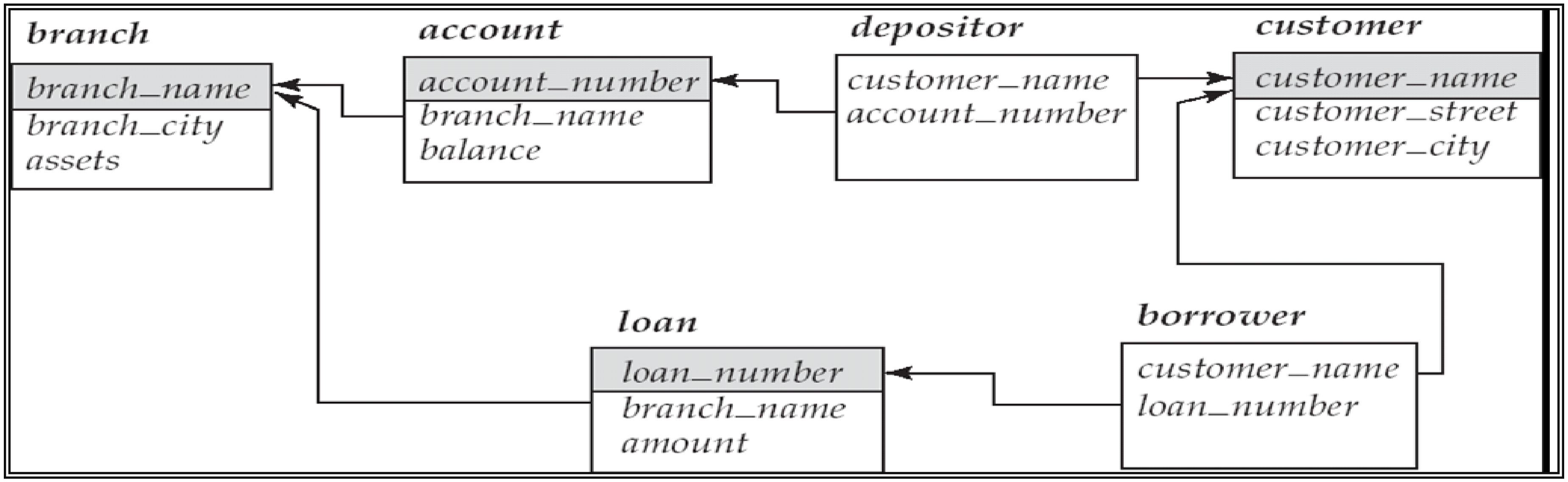
Project Operation

- **Find the names of all customers who have a loan at the Perryridge branch.**

$$\Pi_{customer\_name}\ (\sigma_{branch\_name=\text{“Perryridge”}}\ (\sigma_{borrower.loan\_number\ =\ loan.loan\_number}(borrower \times loan)))$$

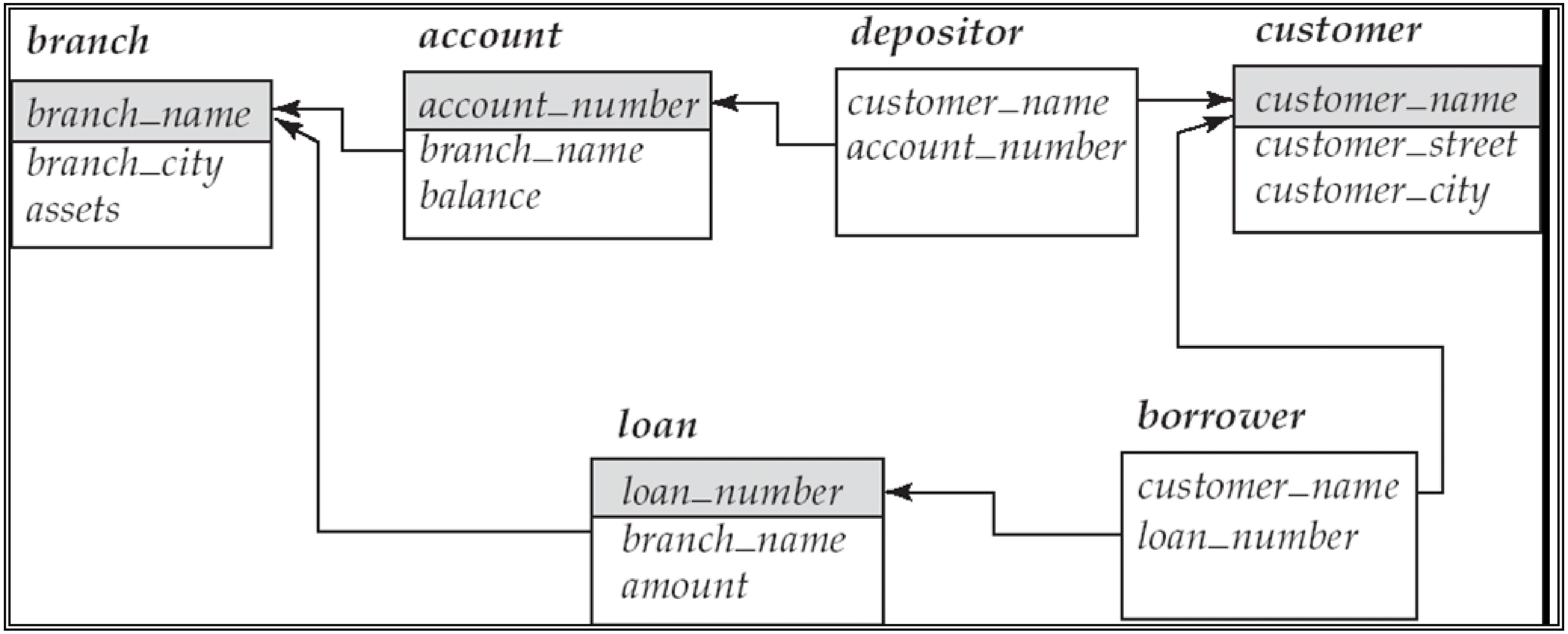
- **Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.**

$$\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{“Perryridge”}}\ (\sigma_{borrower.loan\_number\ =\ loan.loan\_number}(\text{borrower} \times \text{loan}))) - \Pi_{customer\_name}(\text{depositor})$$



Project Operation

- **Find the names of all customers who have a loan at the Perryridge branch**

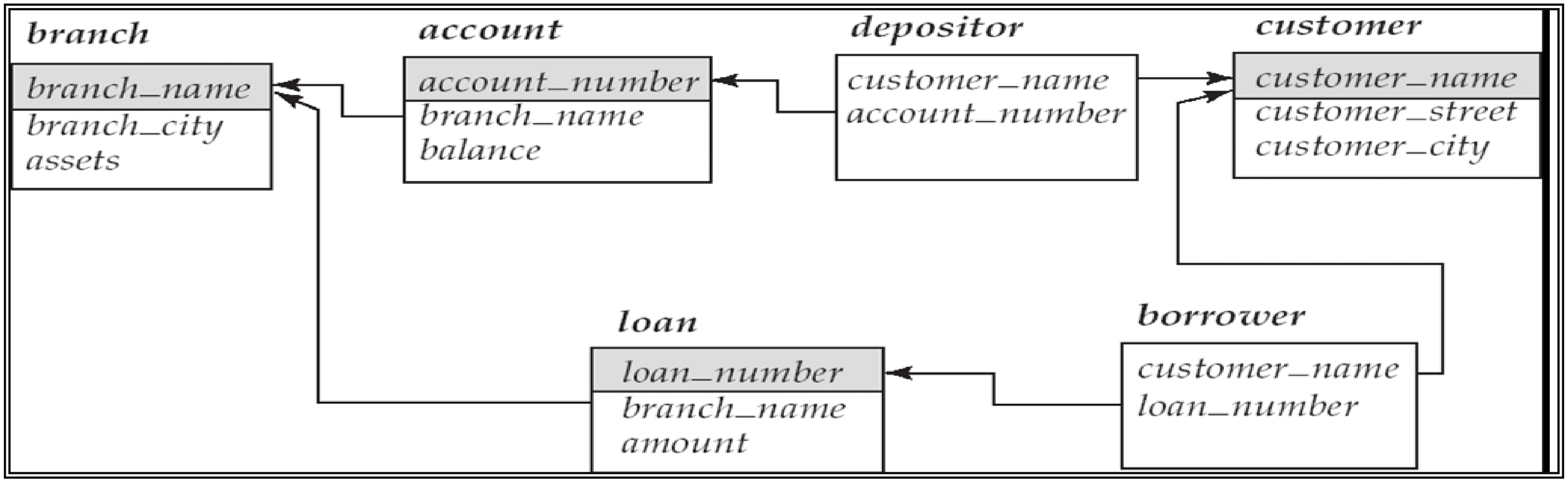


Project Operation

- **Find the names of all customers who have a loan at the Perryridge branch**

- $\Pi_{\text{customer\_name}} (\sigma_{\text{branch\_name = "Perryridge"}} (\sigma_{\text{borrower.loan\_number = loan.loan\_number}} (\text{borrower x loan})))$

- $\Pi_{\text{customer\_name}}(\sigma_{\text{loan.loan\_number = borrower.loan\_number}} ((\sigma_{\text{branch\_name = "Perryridge"}} (\text{loan})) \text{ x } \text{borrower}))$



Set-Intersection Operation

- **Additional Operations**
  - **Set intersection ($\cap$)**
  - **Natural join ($\bowtie$)**
  - **Aggregation**
  - **Outer Join**
  - **Division**

Set-Intersection Operation

- **Relation *r, s*:**

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- ***r* ∩ *s***

A	B
α	2

Natural Join Operation

- Relations 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	$\in$

s

- r ⋈ 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

Natural-Join Operation

- **Notation: r ⋈ s**

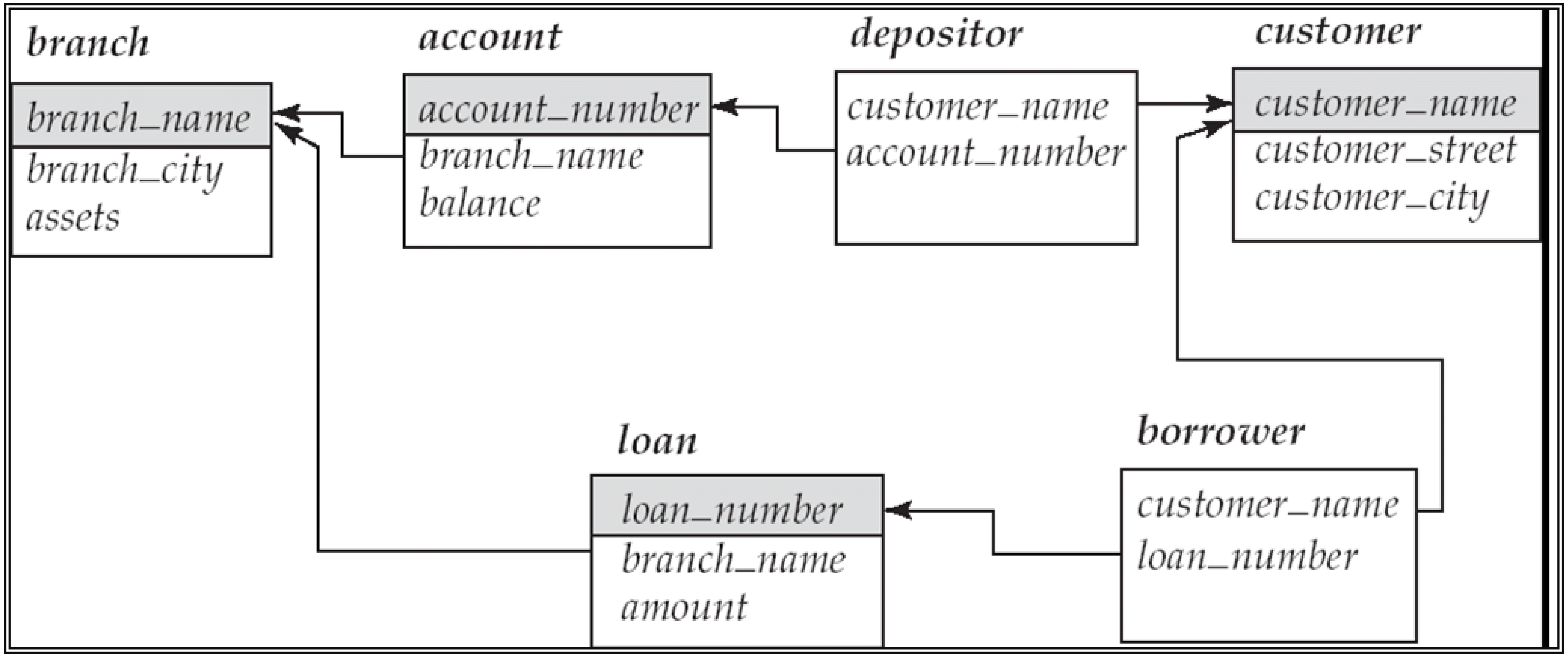
- Let *r* and *s* be relations on schemas *R* and *S* respectively.
  Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*

- Example:
  R = (*A*, *B*, *C*, *D*)
  S = (*E*, *B*, *D*)
  - Result schema = (*A*, *B*, *C*, *D*, *E*)
  - *r* ⋈ *s* is defined as:

$$\prod_{r.A,\ r.B,\ r.C,\ r.D,\ s.E} (\sigma_{r.B = s.B \land r.D = s.D} (r \times s))$$

Natural Join

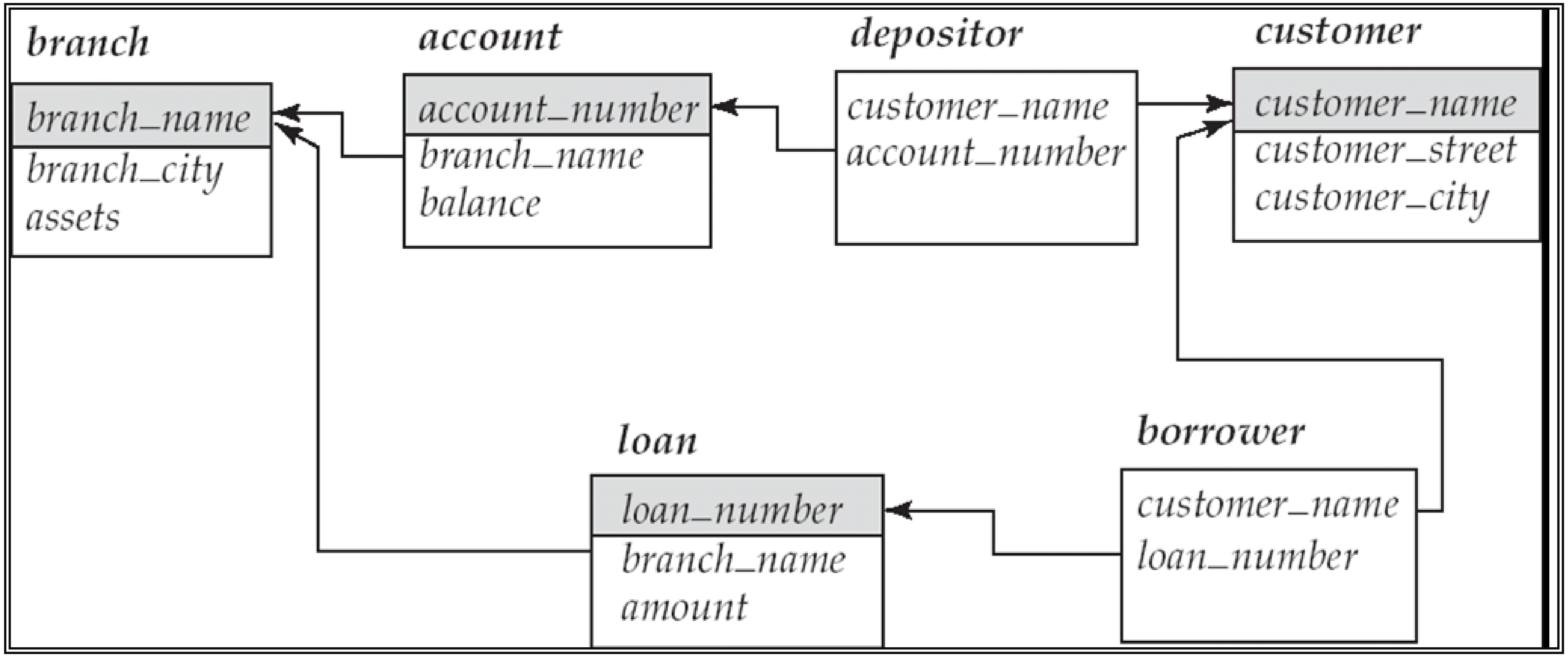
- **Find the name of all customers who have a loan at the bank and the loan amount**



Natural Join

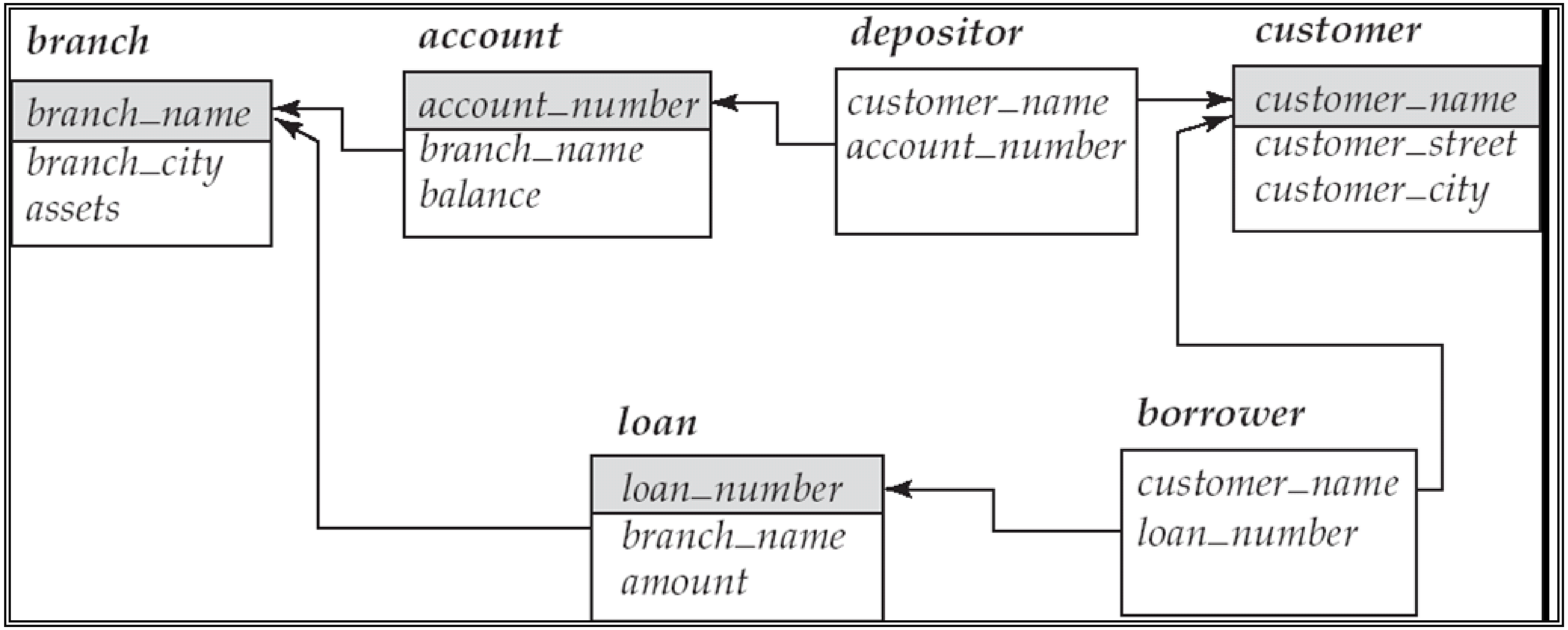
- **Find the name of all customers who have a loan at the bank and the loan amount**

$$\Pi_{customer\_name,\ loan\_number,\ amount}\ (borrower \bowtie loan)$$



Project Operation

- **Find all customers who have an account from at least the "Downtown" and the Uptown" branches.**

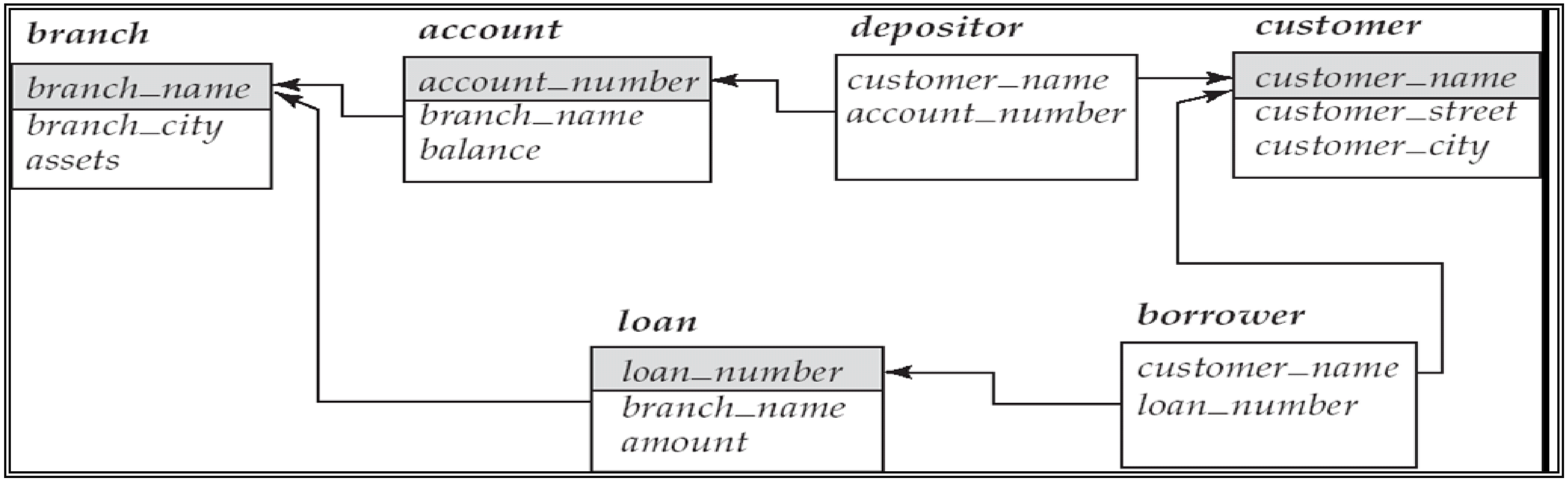


Project Operation

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.

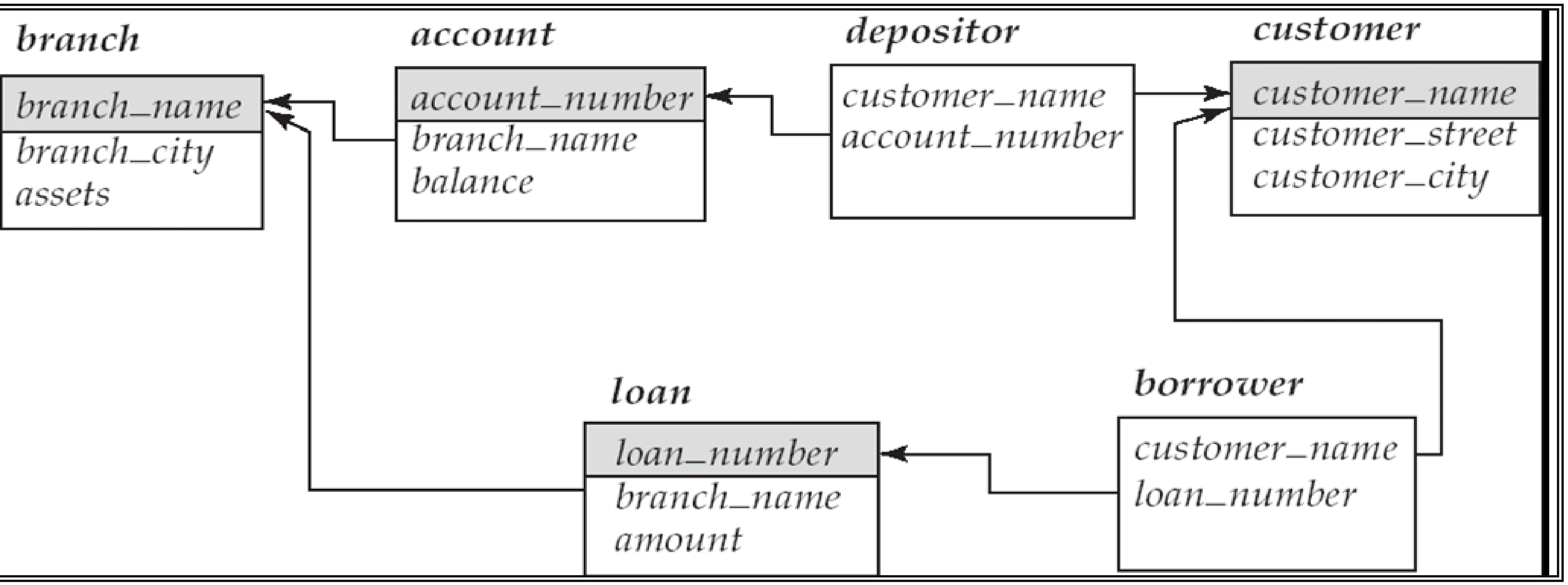
$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{"Downtown"}}(depositor \bowtie account)) \cap$$

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{"Uptown"}}(depositor \bowtie account))$$



Project Operation

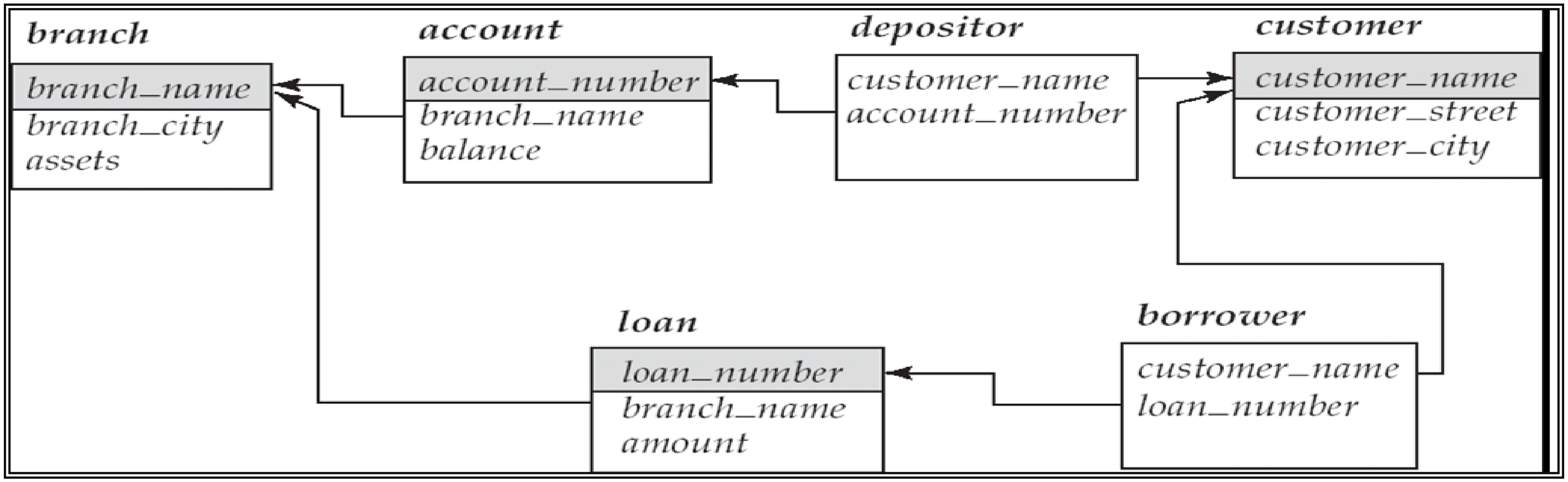
- **Find the largest account balance.**



Project Operation

- **Find the largest account balance**
  - **Strategy:**
    - **Find those balances that are *not* the largest**
      - **Rename *account* relation as *d* so that we can compare each account balance with all others**
    - **Use set difference to find those account balances that were *not* found in the earlier step.**

$$\Pi_{balance}(account) - \Pi_{account.balance}\ (\sigma_{account.balance\ <\ d.balance}\ (account \times \rho_d\ (account)))$$



Aggregate Functions and Operations

- **Aggregate functions that summarize data from tables**

- **Aggregation function takes a collection of values and returns a single value as a result.**
- **avg: average value**
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation in relational algebra**

$$_{G_1,G_2,\ldots,G_n}\vartheta_{F_1(A_1),F_2(A_2,\ldots,F_n(A_n)}(E)$$

***E* is any relational-algebra expression**

- ***G_1, G_2 …, G_n* is a list of attributes on which to group**
- **Each *F_i* is an aggregate function**
- **Each *A_i* is an attribute name**

Aggregate Operation

- **Relation *r*:**

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathcal{g}_{\text{sum(c)}}\text{(r)}$

sum(*c*)
27

Aggregate Operation

- **Relation *account* grouped by *branch-name*:**

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$_{branch\_name}\,\mathcal{G}\,_{\mathbf{sum}(balance)}\,(account)$$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Outer Join

- **An extension of the join operation that avoids loss of information.**
- **Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.**

Example:

Loan

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Borrower

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

- **Join** *loan* $\bowtie$ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- **Left Outer Join** *loan* $⟕$ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Outer Join

- Right Outer Join

  loan $⋊$ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- Full Outer Join

  loan $⟗$ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Null Values

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
  null signifies an unknown value or that a value does not exist.

- The result of any arithmetic expression involving null is null.. Aggregate functions simply ignore null values (as in SQL)

- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL) Comparisons with null values return the special truth value: unknown

- If false was used instead of unknown, then not (A < 5)
  would not be equivalent to A >= 5

- Three-valued logic using the truth value unknown:
  - OR: (unknown or true) = true,
    (unknown or false) = unknown
    (unknown or unknown) = unknown
  - AND: (true and unknown) = unknown,
    (false and unknown) = false,
    (unknown and unknown) = unknown
  - NOT: (not unknown) = unknown

Division Operation

- **Notation:** $r \div s$
- **Suited to queries that include the phrase "for all".**
- **Let *r* and *s* be relations on schemas *R* and *S* respectively where**
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of r ÷ s is a relation on schema

  $R - S = (A_1, \ldots, A_m)$

$$r \div s = \{ t \mid t \in \prod_{R-S}(r) \wedge \forall u \in s \, (tu \in r) \}$$

Where *tu* means the concatenation of tuples *t* and *u* to produce a single tuple

Division Operation

- Relations *r, s*:

A	*B*
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

- $r \div s$:

A
α
β

Division Operation

- Relations *r*, *s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	a	*1*
α	a	γ	b	*1*
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	a	*1*
γ	a	γ	b	*1*
γ	a	β	b	*1*

r

D	*E*
a	*1*
b	*1*

s

- *r* ÷ *s*:

A	*B*	*C*
α	a	γ
γ	a	γ

Division Operation

- **Property**
  - **Let $q = r \div s$**
  - **Then q is the largest relation satisfying $q \times s \subseteq r$**
- **Definition in terms of the basic algebra operation**

Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

$$r \div s = \prod_{R-S} (r) - \prod_{R-S} ((\prod_{R-S} (r) \times s) - \prod_{R-S,S}(r))$$

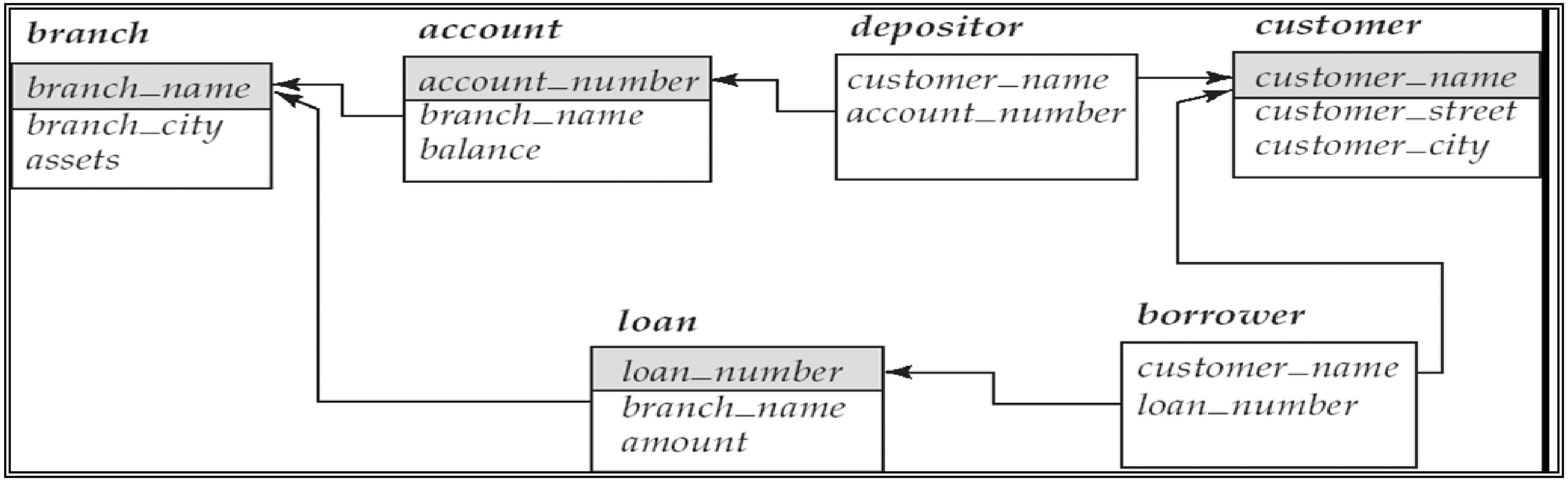
To see why

- $\prod_{R-S,S} (r)$ **simply reorders attributes of r**

- $\prod_{R-S} (\prod_{R-S} (r) \times s) - \prod_{R-S,S}(r))$ **gives those tuples t in**

  $\prod_{R-S} (r)$ **such that for some tuple $u \in s$, $tu \notin r$.**

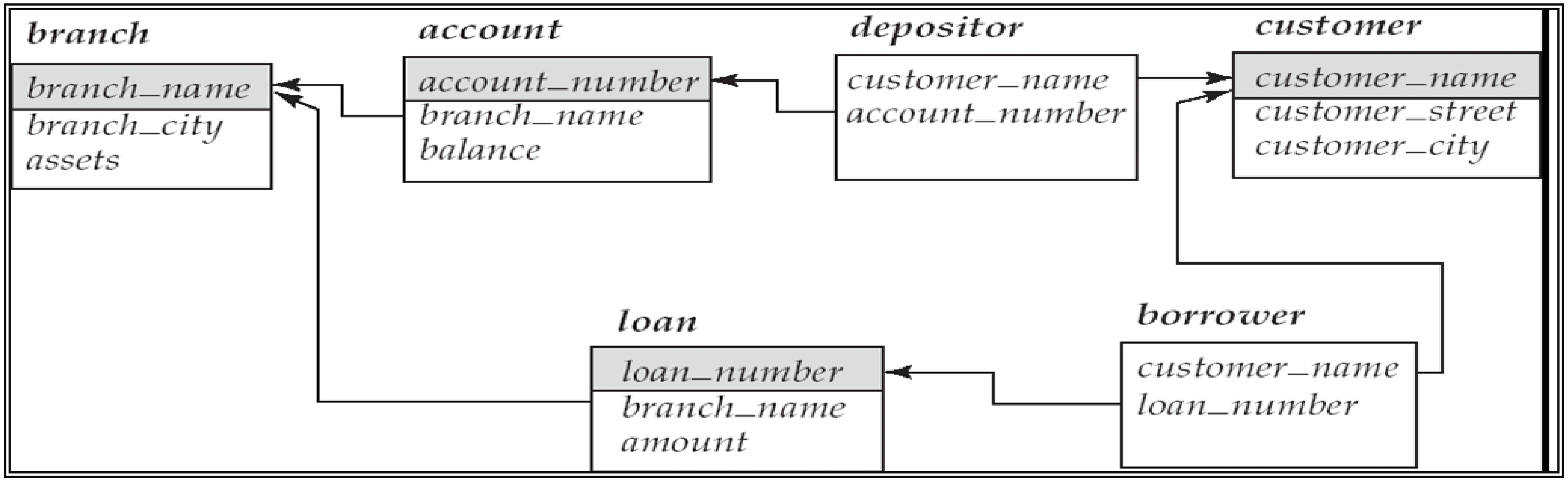
- **Find all customers who have an account at all branches located in Brooklyn city.**



Natural Join and Division

- **Find all customers who have an account at all branches located in Brooklyn city.**

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_city = \text{“Brooklyn”}}(branch))$$

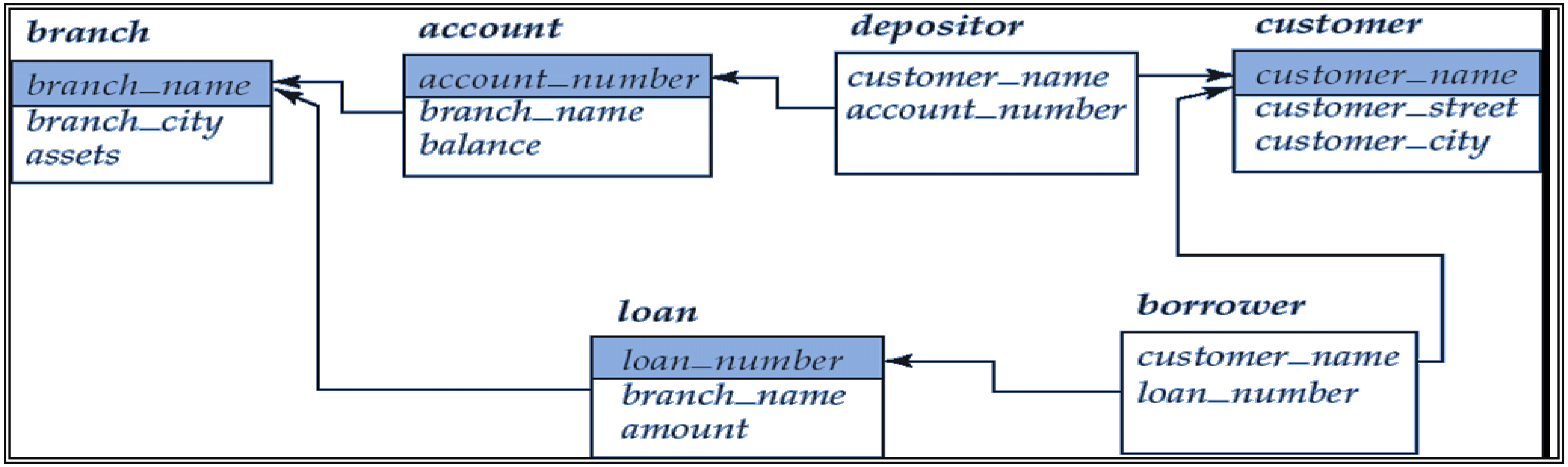


Natural Join

- **Find all customers who have an account from at least the "Downtown" and the Uptown" branches.**
  - $\Pi_{customer\_name} (\sigma_{branch\_name = \text{"Downtown"}} (depositor \bowtie account)) \cap$

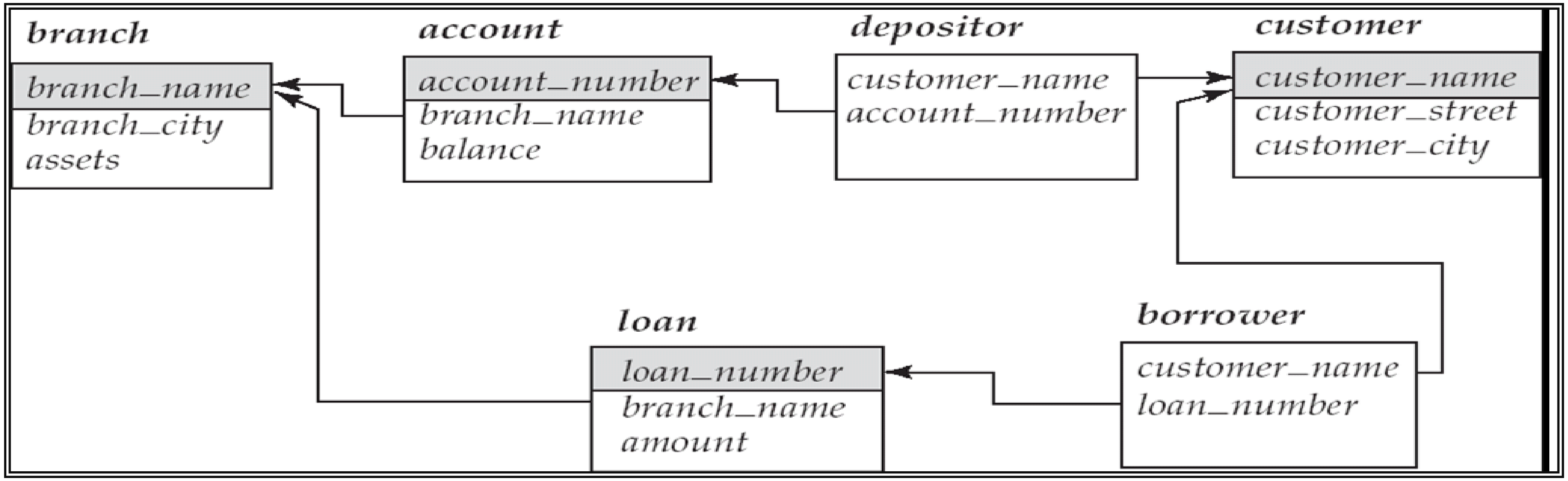
    $\Pi_{customer\_name} (\sigma_{branch\_name = \text{"Uptown"}} (depositor \bowtie account))$

$$\Pi_{customer\_name,\ branch\_name} (depositor \bowtie account)$$
$$\div\ \rho_{temp(branch\_name)} (\{(\text{"Downtown"}), (\text{"Uptown"})\})$$



- **Find all customers who have an account at all branches located in Brooklyn city**

$$\Pi_{customer\_name,\ branch\_name}\ (depositor \bowtie account) \div \Pi_{branch\_name}\ (\sigma_{branch\_city\ =\ \text{“Brooklyn”}}\ (branch))$$



Formal Definition

- **A basic expression in the relational algebra consists of either one of the following:**
  - **A relation in the database**
  - **A constant relation**
- **Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:**
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \text{ x } E_2$
  - $\sigma_p (E_1)$**, *P* is a predicate on attributes in** E_1
  - $\prod_s(E_1)$**, *S* is a list consisting of some of the attributes in** E_1
  - $\rho_x (E_1)$**, x is the new name for the result of** E_1

Select Operation

- **Notation:** $\sigma_p(r)$
- ***p*** **is called the selection predicate**
- **Defined as:**

$$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where *p* is a formula in propositional calculus consisting of **terms**

connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where ***op*** is one of: $=, \neq, >, \geq. <. \leq$

- **Example of selection:**

$$\sigma_{branch\_name=\text{“Perryridge”}}(account)$$

Project Operation

- **Notation:** $\prod_{A_1, A_2, \ldots, A_k} (r)$

  where A_1, A_2 are attribute names and r is a relation name.

- **The result is defined as the relation of k columns obtained by erasing the columns that are not listed**

- **Duplicate rows removed from result, since relations are sets**

- **Example: To eliminate the *branch_name* attribute of *account***

  $$\prod_{account\_number,\ balance} (account)$$

Union Operation

- **Notation: $r \cup s$**
- **Defined as:**

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- **For $r \cup s$ to be valid.**

1. *r*, *s* must have the *same* arity (same number of attributes)

2. The attribute domains must be compatible

(example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

- **Example: To find all customers with either an account or a loan**

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

Set Difference Operation

- **Notation: $r - s$**
- **Defined as:**

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$

- **Set differences must be taken between compatible relations.**
  - ***r* and *s* must have the same arity**
  - **attribute domains of *r* and *s* must be compatible**

Cartesian product

- **Notation: $r \times s$**
- **Defined as:**

$$r \times s = \{t\, q \mid t \in r \text{ and } q \in s\}$$

- **Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).**
- **If attributes of *r* and *s* are not disjoint, then renaming must be used.**