CS & IT College
2020/2021 Semester 1

IS203 Database Principals

Chapter 3: Relational Model

- *Structure of Relational Databases*
- *Relational Algebra*

Asst Prof. Asaad Alhijaj

Reference:

"Database System Concepts Fourth Edition" by Abraham Silberschatz Henry F. Korth S. Sudarshan , McGraw-Hill ISBN 0-07-255481-9

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

Relation Schema

- $A_1, A_2, \ldots, A_n$ are *attributes*
- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  E.g. *Customer-schema* =
  (*customer-name, customer-street, customer-city*)
- $r(R)$ is a *relation* on the *relation schema* R

  E.g. *customer (Customer-schema)*

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** **(لاتتجزأ)**, that is, indivisible
  - E.g. **multivalued** attribute values are not atomic
  - E.g. **composite** attribute values are not atomic
- The special value ***null*** is a member of every domain
- The null value causes complications in the definition of many operations
  - we shall ignore the effect of null values in our main presentation and consider their effect later

Basic Structure

- Formally, given sets D_1, D_2, …. D_n a **relation** r is a subset of $D_1 \times D_2 \times \ldots \times D_n$
  Thus a relation is a set of n-tuples (a_1, a_2, …, a_n) where each $a_i \in D_i$
- Example: if

  customer-name = {Jones, Smith, Curry, Lindsay}
  customer-street = {Main, North, Park}
  customer-city = {Harrison, Rye, Pittsfield}

  Then *r* = { (Jones, Main, Harrison),
  (Smith, North, Rye),
  (Curry, North, Rye),
  (Lindsay, Park, Pittsfield)}

  is a relation over *customer-name x customer-street x customer-city*

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* : stores information about accounts
  depositor : stores information about which customer owns which account
  customer : stores information about customers

- Storing all information as a single relation such as
  bank(*account-number, balance, customer-name*, ..)
  results in
  - repetition of information (e.g. two customers own an account)
  - the need for null values (e.g. represent a customer without an account)
- Normalization theory (Chapter 7) deals with how to design relational schemas

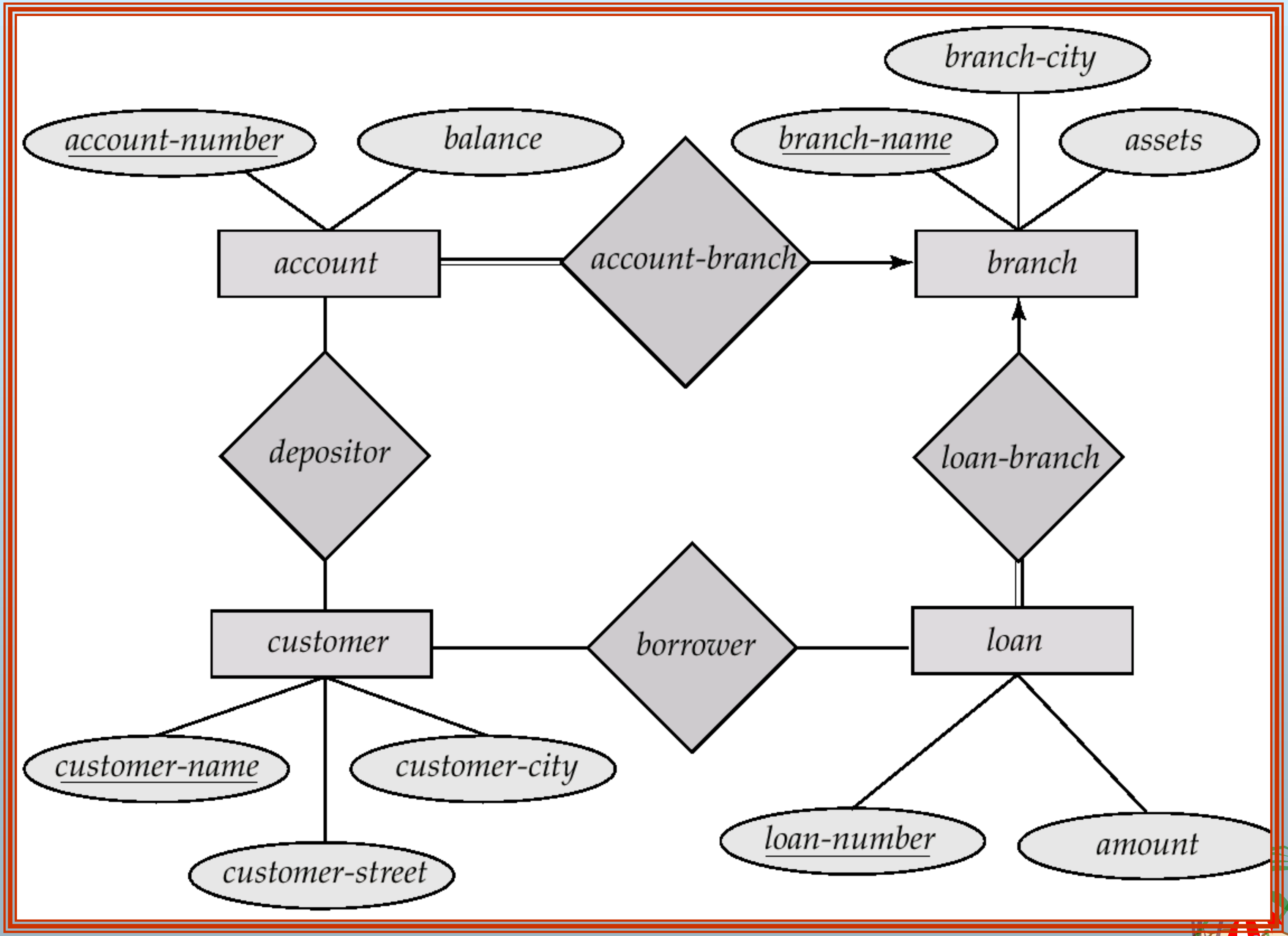
The *customer* Relation

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

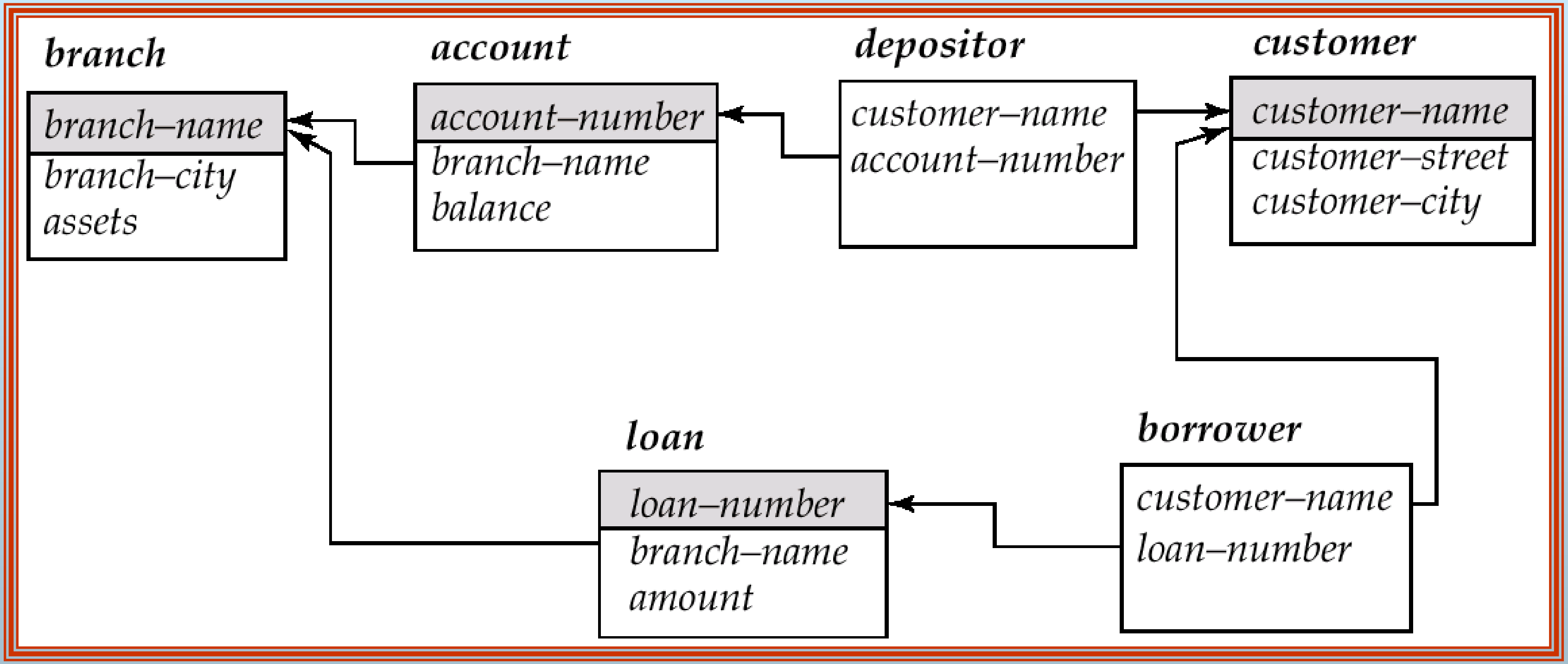
The *depositor* Relation

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

E-R Diagram for the Banking Enterprise



Schema Diagram(UML) for the Banking Enterprise



Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - procedural
  - non-procedural
- "Pure" languages:
  - **Relational Algebra**
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Pure languages form underlying basis of query languages that people use.

Relational Algebra

- Procedural language
- Six basic operators :
  - select
  - project
  - union
  - set difference
  - Cartesian product
  - rename
- The operators take one or more relations as *inputs* and give a new relation as a *result*.

Select Operation – Example

- 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

Select Operation

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

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

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:

$\sigma_{\textit{branch-name="Perryridge"}}(\textit{account})$

Project Operation – Example

- 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

Project Operation

- Notation:

$$\Pi_{A1,\ A2,\ \ldots,\ Ak}\ (r)$$

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
- E.g. To eliminate the *branch-name* attribute of *account*

$$\Pi_{account\text{-}number,\ balance}\ (account)$$

Union Operation – Example

- Relations *r, s:*

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

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* (e.g., 2nd column of r deals with the same type of values as does the 2nd column of s)

- E.g. to find all customers with either an account or a loan

$$\Pi_{customer\text{-}name}\ (depositor) \cup \Pi_{customer\text{-}name}\ (borrower)$$

Set Difference Operation – Example

- 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

Set Difference Operation

- Notation $r - s$
- Defined as:

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

- 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 Operation-Example

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

Cartesian-Product Operation

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

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

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

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}\,(r \times s)$
- *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

- $\sigma_{A=C}(r \times s)$

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

$$\rho_{X\,(A1,\,A2,\,\ldots,\,An)}(E)$$

returns the result of expression *E* under the name *X*, and with the

attributes renamed to *A1, A2, …., An*.

Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Example Queries

- Find all loans of over $1200

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

- Find the loan number for each loan of an amount greater than $1200

$$\Pi_{loan\text{-}number}\ (\sigma_{amount\ >\ 1200}\ (loan))$$

Example Queries

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

$$\Pi_{\textit{customer-name}}\ (\textit{borrower}) \cup \Pi_{\textit{customer-name}}\ (\textit{depositor})$$

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{\textit{customer-name}}\ (\textit{borrower}) \cap \Pi_{\textit{customer-name}}\ (\textit{depositor})$$

Example Queries

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

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Perryridge”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$$

- 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.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(\text{borrower x loan})))\ -\ \Pi_{customer\text{-}name}(\text{depositor})$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

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

  - Query 2

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

Example Queries

Find the largest account balance

- Rename *account* relation as *d*
- The query is:

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