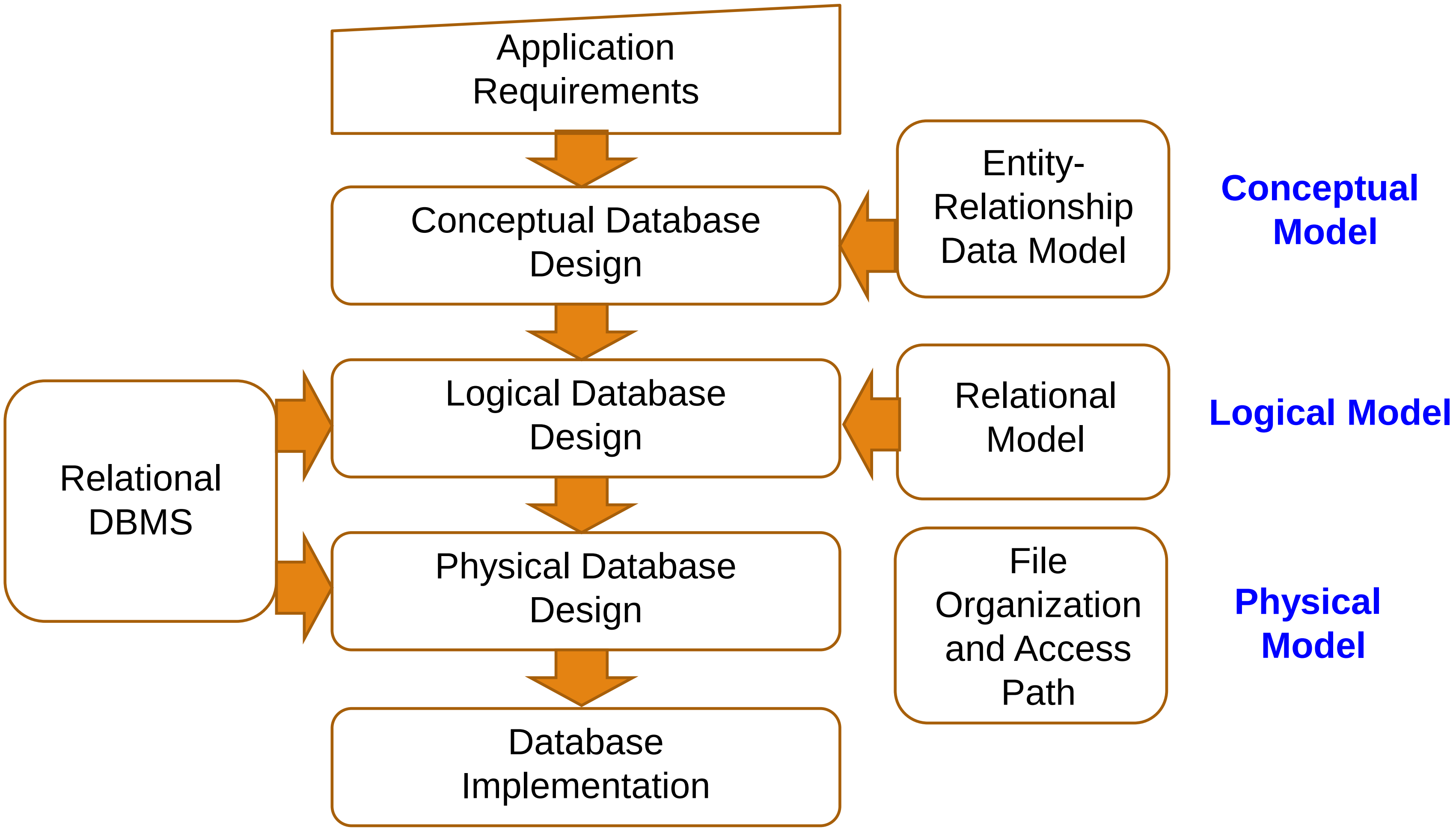
Lecture 2: Introduction to Relational Model and Fundamental Relational Algebra Operators

CS2204/CSX3006 DATABASE SYSTEMS

ITX3006 DATABASE MANAGEMENT SYSTEMS

Big Picture: Design Phases - 1



Big Picture: Design Phases - 2

- Conceptual Model
  - Captures the **requirements** and **rules** of an application in terms of semantics of required **entities**, **relationships** among the entities and consistency **constraints**.
  - Entity-Relationship Model
- Logical Model
- Physical Model

Big Picture: Design Phases - 3

- Conceptual Model
- Logical Model
  - Specifying **particular data structures** to represent the **entities**, **relationships** and **constraints** defined in the conceptual model
    - Facilitate manipulation and retrieval of data stored in the database
  - Relational Database Model and Structured Query Language (SQL):
    - Tables, keys and constraints
    - Specification of structures and constraints → DDL
    - Manipulation and Retrieval of data → DML
      - Exact syntax and semantics vary from implementation to implementation (different DBMS products), but
      - All are based on the SAME mathematical concept of '**relation**' and '**relational algebra**'
- Physical Model

Big Picture: Design Phases - 4

- Conceptual Model
- Logical Model
- Physical Model
  - Internal data storage structuring and organization of DBMS

Objectives and Outline

- Objectives
  - Understand the structure of relational database
  - Be able to express queries using relational algebra

- Relational Database Model
  - Structure of Relational Database
    - table, column, row
    - relation, attribute, domain and tuple
  - Data Consistency Constraints
    - Unique Tuple Identification: Super key, Candidate Key, Primary Key
    - Referential Integrity and Foreign Keys
  - Fundamental Relational Algebra Operators
    - select: σ, project: $\prod$, union: $\cup$, set difference: –, Cartesian product: x, rename: ρ

Why Study the Relational Database Model?

- Most widely used model; Logical Model of most DBMS
  - Oracle, MS SQL, IBM DB2, PostgreSQL, Microsoft SQL Server, ...
- Very Simple yet Extremely Useful
  - Single Data modelling concept: **relation** (a.k.a. **table**)
- Allows clean yet powerful manipulation language

Basic Structure of Relational Database - 1

- A **Relational Database: a collection of relations** (tables), each with a unique name
- A **table**: 2 dimensional structure, consisting of
  - **Column (field, attribute)**
    - Each **attribute** has *a set of permitted values* → **domain** of the attributes
      - In pure relational database, *domain values must be **atomic.***
        - *Each value is indivisible simple value, e.g., age, first name, balance*
  - **Row (record, instance, tuple)**

Example of a Relation (Table)

Column headers

Attribute names

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

a row

a tuple

account table

***account* relation**

Basic Structure of Relational Database - 2

- A Relation: a set of records (tuples)
  - ORDER of records does NOT matter.
  - **Each record** has to be **uniquely distinguishable** from others in a relation.
    - No multiple copies of the same tuple in a set

Domain of Attribute

Attribute names

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

- **Domain**: A set of permitted values for an attribute
  - E.g., Domain of the branch_name is the set of all branch names
- **Logical Level Domain:** A set of all branch names
- **Physical Level Domain:** char(30); character string consisting of 30 characters

Question: What are the Logical (Physical) Level domains for account_number and balance?

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

- Specification of Domain is based on required Business Logic!

Domain of Attribute - 1

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 relation

employee_name	*branch_name*	*salary*
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

employee relation

- Should **customer_name** and **employee_name** have same **Logical Level** domain (**Physical Level** domain)?

Domain of Attribute - 2

employee_name	*branch_name*	*salary*
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

employee relation

- Should **employee_name** and **branch_name** have same **Logical Level** domain (**Physical Level** domain)?

A Null Value

- Is a member of any possible domain
- Can signify "**unknown**" or "**not exist**" or "**not sure**"
- In theory, good designs should try to *avoid null values* if possible
- In practise, null values are used to *avoid 'too complex'* design

customer_name	customer_street	customer_city	customer_phone
Adams	Spring	Pittsfield	091234567
Curry	North	Rye	**null**
Green	Walnut	Stamford	076752345

Formal Definition of Relation

- Given **sets** D_1, D_2, …. D_n, representing **Domains,**
  - a **relation** *r* is
    - **a subset of** $D_1 \times D_2 \times \ldots \times D_n$ (**Cartesian Product of *n* Sets**)
    - **a set of *n*-tuples ($a_1, a_2, \ldots, a_n$) where each $a_i \in D_i$**
      - ***D_1, D_2, …. D_n : domains of attributes***
      - ***a_1, a_2, …, a_n : attribute values** in each respective domains*
      - *a **particular instance** of* (a_1, a_2, …, a_n): ***a tuple***

- A relation is a set of such tuples that *satisfies a certain 'property'* defining connection among attribute values of a n-tuple.

Example of Relation Customer

- If
  - *customer_name* = {Jones, Smith, Curry, Lindsay, …} /* Set of all customer names */
  - *customer_street* = {Main, North, Park, …} /* set of all street names*/
  - *customer_city* = {Harrison, Rye, Pittsfield, …} /* set of all city names */

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

is a relation over

customer_name × *customer_street* × *customer_city*

Relation Schema - 1

customer

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

The **schema** of a relation

- **Schema** defines the structure of the relation;
  - Definition of the relation + Integrity Constraints + Keys, etc.

Relation Schema - 2

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

  Example:

  Customer_schema = (*customer_name, customer_street, customer_city*)

- *r*(*R*) denotes a *relation r* on the *relation schema R*

  Example: *customer (Customer_schema)*

  "customer is a relation conforming to Customer_schema"

Relation Instance

customer **At time t1**

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

An instance of the relation

The **current** set of tuples in a relation

customer **At time t2**

customer_name	*customer_street*	customer_city
Jones	Main	Harrison
Smith	North	Rye
Turner	**Putnam**	**Stamford**
Lindsay	Park	Pittsfield
Green	**Walnut**	**Stamford**
Williams	**Nassau**	**Princeton**

Another instance of the relation

Relations are Unordered

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

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

- Are they the same instance?

Relational Database - 1

- A relational database **consists of one or** (typically) **more 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

See also: Relational Model of the Banking Enterprise DB.pdf

Relational Database - 2

- Why break up into multiple relations? Why not single relation having everything?
  - Storing all information as a single relation such as
    bank(*account_number, branch, balance, customer_name*, ... , ..., ..., ...)
    results in
    - repetition of information (Data Redundancy → Data Inconsistency)
    - Inability to represent information; the need for null values
  - Normalization theory deals with how to design good relational schemas

A simple Relational Database example

Customer_schema = (customer_name, customer_street, customer_city)

Account_schema=(account_number, branch_name, balance)

Depositor_schema=(customer_name, account_number)

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

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

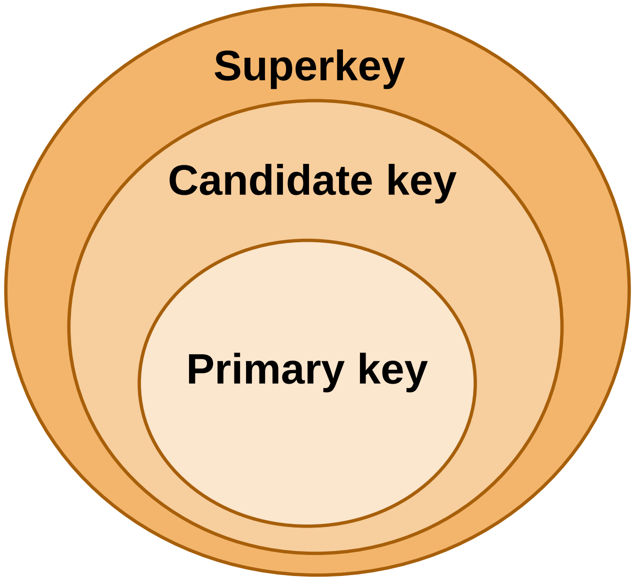
Recall the Fact about Relation

- A relation is a set of tuples
- Each tuple in a relation must be uniquely distinguishable from others (→ different **attribute values**)
  - **No** two tuples in a relation have exactly the same values for all attributes → A **Key** is needed.

Consider bank customers have deposit balances.

Can we use their name as a **key**?

Different Types of Keys and their Relationships



Superkey

- A **superkey**: a set of *one or more* **attributes** that allow us **to identify uniquely a tuple** in the relation.
- Let K ⊆ R; (R is a relation schema; the set of attribute names for the relation)
- *K* is a superkey of *R* **if** values for *K* are **uniquely identify each tuple** of each possible relation *r(R)*
- Example:
  - {*customer_name, customer_street*}, and
  - {*customer_name, customer_street, customer_city*}

  are a superkeys of *Customer*, IF NO two customers living on the same street can possibly have the SAME name

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 set of all attributes is always a superkey **(Trivial superkey).**

Candidate Key

- *K* is a **candidate key** if *K* is a minimal superkey
- Example: candidate keys for Customer
  - {*mobile_phone*}
  - {customer_name, customer_street}
    - Under the assumption of the business logic discussed in the previous slide

Note. This column is added for demonstration

customer_name	customer_street	customer_city	mobile_phone
Adams	Spring	Pittsfield	7154628
Brooks	Senator	Brooklyn	2548621
Curry	North	Rye	5841254
Glenn	Sand Hill	Woodside	2554478
Green	Walnut	Stamford	1236545
Hayes	Main	Harrison	9854123
Johnson	Alma	Palo Alto	9874100
Jones	**Main**	Harrison	9568323
Lindsay	Park	Pittsfield	9786541
Smith	North	Rye	6958569
Turner	Putnam	Stamford	2123021
Williams	Nassau	Princeton	9851459
Jones	**North**	Princeton	5114233

Primary Key

- Primary key (PK): **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

What are Primary Keys in this Relational Database? - 1

Customer_schema = (customer_name, customer_street, customer_city)

Account_schema=(account_number, branch_name, balance)

Depositor_schema=(customer_name, account_number)

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

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

What are Primary Keys in this Relational Database? - 2

Customer_schema = (<u>customer_name</u>, customer_street,

customer_city)

Account_schema=(<u>account_number</u>, branch_name, balance)

Depositor_schema=(<u>customer_name, account_number</u>)

***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

***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

***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

Database Constraints vs Business Logic

- **Schema of Database also represents "constraints"** that the data in the database must followed.
- These "**constraints**" are **means of specifying certain** (but not all) **Business Logic**
- e.g.) **Domain Specification for an Attribute**

Attribute	**Domain**
customer_age	Integer Value from 1 to 199

Basic Constraints in Database

- Domain constraints
- Key constraints
- Referential Integrity constraint (Foreign key constraint)

Key Specification (Constraint)

- **Key Specification** (esp. **Primary Key** and **Candidate Keys**) is a kind of **constraint**
- Choice of candidate keys and the primary key of relations must be made based on **Business Logic** and **Application Requirements**
  - **Depositor_schema = (<u>customer_name, account_number</u>) or Depositor_schema = (customer_name, <u>account_number</u>) ?**
- In real applications, we tend to use some "artificial sequencing mechanism" as the primary key of relations

  Need to pay also attention to other candidate keys that may exist in your relation schema since they represent business rules that must be enforced!

***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

Foreign Key and Referential Integrity

- **Foreign key (FK): an attribute** in some relation schema **that corresponds to the primary (or candidate) key of another relation**.
- **Referential Integrity**
  - A property of data that only values occurring in the Key attributes of the **referenced relation** may occur in the foreign key attribute of the **referencing relation.**

PK

Customer_schema = (customer_name, customer_street, customer_city)

PK

Account_schema=(account_number, branch_name, balance)

Depositor_schema=(customer_name, account_number)

FK FK

Note: the attributes at the tail of arrows are **FK**.

Question

- According to **Referential Integrity constraint,** can we add the following tuple to the Depositor relation?
  - ("Alex", A-103); to reflect the fact that Customer named "Alex" has opened account A-103

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

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

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

Representing Database Schema

Branch_schema = (branch_name, branch_city, assets)

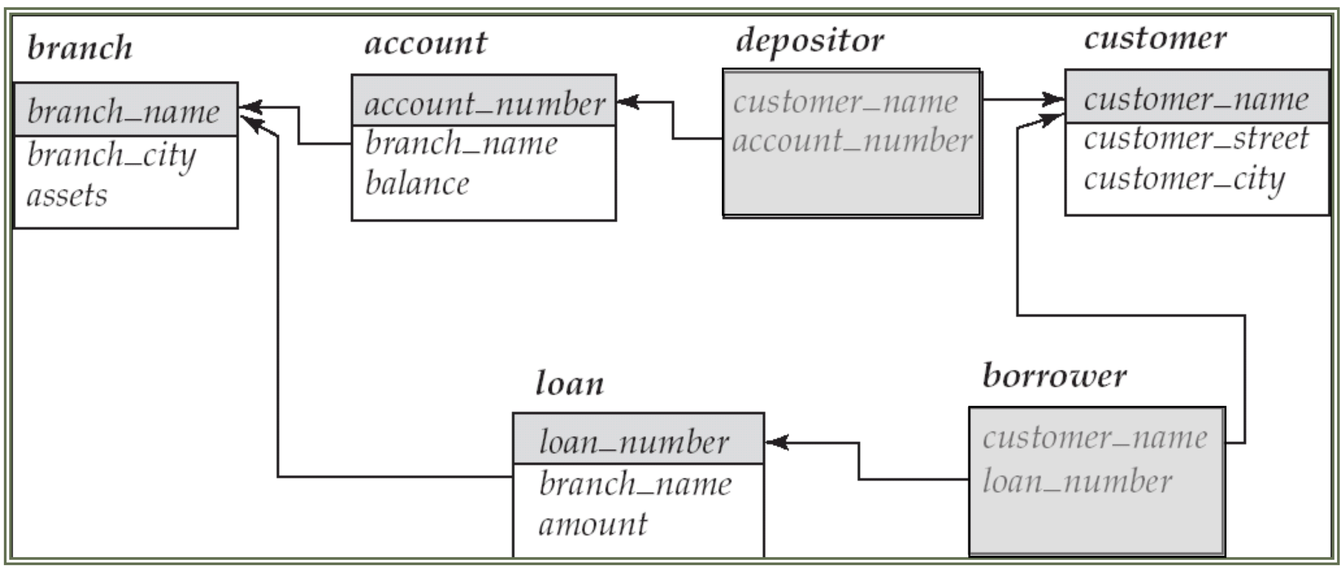
Account_schema = (account_number, *branch_name*, balance)

Depositor_schema = (*customer_name*, *account_number*)

Customer_schema = (customer_name, customer_street, customer_city)

Loan_schema = (loan_number, *branch_name*, amount)

Borrower_schema = (*customer_name*, *loan_number*)



Typical Steps in Defining Relational Database Schema

1. **Define the relations**, with a unique name for each relation
2. **Define attributes** for each relation and specify the domain for each attribute
3. **Specify the key(s)** for each relation
   - Candidate keys and choose a primary key
4. **Specify** all appropriate **foreign keys and integrity constraints**

Another Simple Example - 1

Teacher = (<u>Number</u>, Name, Office, E-mail)

Course = (<u>Number</u>, Name, Credit, Description)

Class = (Semester, Course_ID, Section, TimeDays, Room)

Taught-by = (Teacher, Semester, Course, Section, Performance)

- What do you think is the purpose of the '**Class**' relation?
  - What should be the **primary key** of the relation? WHY?

Another Simple Example - 2

Teacher = (<u>Number</u>, Name, Office, E-mail)

Course = (<u>Number</u>, Name, Credit, Description)

Class = (Semester, Course_ID, Section, TimeDays, Room)

Taught-by = (Teacher, Semester, Course, Section, Performance)

- Can you figure out the purpose of '**Taught-by**' relation?
  - Can you see what should be the **primary key** for the relation?

Another Simple Example - 3

Teacher = (<u>Number</u>, Name, Office, E-mail)

Course = (<u>Number</u>, Name, Credit, Description)

Class = (Semester, Course_ID, Section, TimeDays, Room)

Taught-by = (Teacher, Semester, Course, Section, Performance)

- What are the **foreign keys** of each relation?

Another Simple Example - 4

Teacher = (Number, Name, Office, E-mail)

Course = (Number, Name, Credit, Description)

Class = (Semester, Course_ID, Section, TimeDays, Room)

Taught-by = (Teacher, Semester, Course, Section, Performance)

Todo Task: Do Worksheet 2 (15 minutes)

Query Languages

- **Used to retrieve information and manipulate data** stored in the relational database in convenient and efficient manner
  - SQL (Structured Query Language) ← the standard way
    - Based on formal mathematical foundations
      - Relational Algebra
      - Relational Calculus

Relational database model supports query language!

Basics of Algebra

- Algebra: the study of mathematical **symbols** and the **rules** for manipulating these symbols in arithmetic expressions
  - Operands: constants and variables
  - Operators: +, –, *, /

- Expressions can be constructed by applying operators to operands and/or other expressions:
  - E.g., (x + y) * 3 / ((x + z – k) * (-y + z))

  - Pipelining: output of one operation is used as an input to another operation

Relational Algebra

- **Operands**: variables that stand for **relation instances** (**a set of tuples**)
- **Operators**:
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- Examples:
  - $\sigma_{amount > 1200}\ (loan)$
  - $\Pi_{customer\_name}\ (depositor)\ \cup\ \Pi_{customer\_name}\ (borrower)$
  - $\Pi_{\text{customer\_name}}\ (\sigma_{\text{branch\_name = “Perryridge”}}\ (\sigma_{\text{borrower.loan\_number = loan.loan\_number}}\ (\text{borrower x loan})))$

Additional Relational Algebra Operators

- **Additional Operators** (Add no power to the fundamental ones, but provides convenience of expressing complex expressions.) → Next Week
  - set intersection: $\cap$
  - natural join: $\bowtie$
  - division: $\div$
  - assignment: $\leftarrow$
- **Extended Operators** (Add more power) → Next Week
  - Generalized Project to allow arithmetic functions in the projection list
  - Aggregate Functions
  - Outer Join
- **Modification Operations** → Next Week
  - How to express deletion, insertion and updating of tuples

In this lecture, we will use **relational algebra** to demonstrate **data retrieval** from **relational database**.

Later on, we will use **SQL** to demonstrate **data retrieval** from **relational database**.

Select Operation - 1

- Notation: $\sigma_p(r)$

σ is pronounced as 'Sigma'

  - Unary operator
  - *p* is called the **selection predicate**
  - Produces a new relation with the subset of the tuples in *r* that **match the predicate *p***
  - *Examples:*
    - $\sigma_{branch\_name=\text{“Perryridge”}}(account)$
    - $\sigma_{amount>1200}(loan)$

Select Operation - 2

- Defined as:

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

  where p is a formula consisting of terms connected by :

  $$\wedge \textbf{ (and)}, \vee \textbf{ (or)}, \neg \textbf{ (not)}$$

  Each term is one of:

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

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

- Meaning: select tuples that satisfy given predicate

- Example of selection:

  $$\sigma_{branch\_name=\text{“Perryridge”} \wedge amount > 1300}(\text{loan})$$

r : relation instances
t : tuple

Select Operation Example - 1

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

loan relation

Q1: Select all loan amounts at the branch ‘Perryridge’

Select Operation Example - 2

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

loan relation

Q1: Select all loan amounts at the branch 'Perryridge'

$\sigma_{\textit{branch\_name}=\text{"Perryridge"}}$ **(loan)**

What's the result of the above query?

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

Remember the result of a relational algebra operator is a relation instance

Select Operation Example - 3

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

loan relation

Q1: Select all loan amounts at the branch ‘Perryridge’ whose amount is greater than 1300.

Select Operation Example - 4

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

loan relation

Q1: Select all loan amounts at the branch 'Perryridge' whose amount is greater than 1300.

$\sigma_{\textit{branch\_name}=\text{"Perryridge"} \wedge \textit{amount} > 1300}$ **(loan)**

loan_number	branch_name	amount
L-15	Perryridge	1500

Select Operation Example - 1

- Relation r

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

- $\sigma_{A=B}(r)$

Select Operation Example - 2

- Relation r

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

- $\sigma_{A=B}(r)$

A	B	C	D
α	α	1	7
β	β	12	3
β	β	23	10

Select Operation Example - 3

- Relation r

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

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

Select Operation Example - 4

- 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

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

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

  - Unary Operator
  - **Produce the relation of *k* attributes obtained by erasing the attributes that are NOT listed**
  - **Duplicate rows removed from result**, **since relations are sets**
  - Example: To eliminate the *branch_name* attribute of *account*

$$\prod_{loan\_number,\ amount}(loan)$$

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

loan_number	*amount*
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Project Operation Example - 1

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

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

Project Operation Example - 2

- 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

A relation is a SET of tuples!

Union Operation - 1

- Notation: $r \cup s$
  - Binary operator
  - The usual set union operation
  - Produce a new relation **containing tuples from *r* and *s* eliminating duplicate tuples**
- Defined as:

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

Union Operation - 2

- 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*

Union Operation - 3

- Example: to find all customers with either a deposit 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	*loan_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

Union Operation - 4

- Example: to find all customers with either an account or a loan (or both)

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

***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	*loan_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

customer_name
Hayes
Johnson
Jones
Lindsay
Smith
Turner
Adams
Curry
Jackson
Williams

Union Operation Example - 1

- Relations *r, s:*

A	*B*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

- r $\cup$ s:

Union Operation Example - 2

- 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

- Notation $r - s$
  - Binary Operator
  - Produce a new relation **consisting of tuples that are in *r BUT NOT* in *s***
  - Defined as:

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

- Set differences must be taken between **compatible** relations

1. *r* and *s* must have the same arity
2. attribute domains of *r* and *s* must be compatible

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

What does the above query find?

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

Cartesian-Product Operation - 1

- Notation $r \times s$
  - Binary Operator
  - Produce a relation that **combines** information from any two relations.
- Defined as: $r \times s = \{ (t\ q) \mid t \in r \textbf{ and } q \in s\}$
  set of all possible **"ordered pairs"** whose first component is a member of *r*, and the second component is a member of *s*
- Assume that **attributes** of r(R) and s(S) are **disjoint**. ($R \cap S = \emptyset$)
  - If attributes of *r(R)* and *s(S)* are **NOT disjoint**, then **renaming** must be used.

Cartesian-Product Operation - 2

- borrower = (customer_name, loan_number)
- loan = (loan_number, branch_name, amount)

- What's the relation schema for borrower x loan ?

A. (borrower.customer_name, borrower.loan_number, loan.loan_number, loan.branch_name, loan.amount)

B. (customer_name, borrower.loan_number,

loan.loan_number, branch_name, amount)

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

Assume we have

n1 tuples in r1 and

n2 tuples in r2.

What's the number of tuples in r3 = r1 x r2?

Cartesian-Product Operation – Example - 1

customer_name	*loan_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

borrower

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

loan

- How can we find the names of all customers who have a loan at the Perryridge branch?
  - Hint: use all operations discussed so far

Cartesian-Product Operation – Example - 2

customer_name	loan_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

borrower

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

loan

How can we find the names of all customers who have a loan at the Perryridge branch?

$\sigma_{branch\_name=\text{“Perryridge”}}(\text{borrower x loan})$?

$\sigma_{borrower.loan\_number = loan.loan\_number}$

$(\sigma_{branch\_name=\text{“Perryridge”}}(\text{borrower x loan}))$?

$\Pi_{\text{customer\_name}}(\sigma_{borrower.loan\_number = loan.loan\_number}$

$(\sigma_{branch\_name=\text{“Perryridge”}}(\text{borrower x loan})))$

Rename Operation

- **The results** of relational algebra expressions **do not have a name**, we may need to **refer to the same relation by more than one name**

- Notations

ρ is pronounced as 'Rho'

- $\rho_x(E)$
  - **returns the expression *E* under the name *X***
- $\rho_{x(A_1, A_2, \ldots, A_n)}(E)$
  - **returns the result of expression *E* under the name *X*, and with the attributes renamed to A_1, A_2, …., A_n.**

Rename Operation Example - 1

- Example: Find the names of all customers who live on the same street and in the same city as Hayes

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	Alma St.	Palo Alto
019-28-3746	Smith	North St.	Rye
677-89-9011	Hayes	Main St.	Harrison
182-73-6091	Turner	Putnam Ave.	Stamford
321-12-3123	Jones	Main St.	Harrison
336-66-9999	Lindsay	Park Ave.	Pittsfield
019-28-3746	Smith	North St.	Rye

(a) The *customer* table

Rename Operation Example - 2

Step 1. Find the street and city of Hayes

$\Pi_{\text{customer\_street, customer\_city}}\ (\sigma_{\textit{customer\_name} = \textit{“Hayes”}}(\text{customer})\ )$

customer-street	**customer-city**
Main St.	Harrison

Rename Operation Example - 3

Step 2. Rename attributes: street and city (of Hayes)

$\rho_{\text{hayes\_addr(hayes\_street, hayes\_city)}} \Pi_{\text{customer\_street, customer\_city}} (\sigma_{\text{customer\_name} = \text{“Hayes”}}$ (customer))

hayes_addr

hayes_street	hayes_city
Main St.	Harrison

Rename Operation Example - 4

Step 3. Concatenate all customer info. with Hayes' street and city.

customer x $(\rho_{\textit{hayes\_addr(hayes\_street, hayes\_city)}}\ \Pi_{\text{customer\_street, customer\_city}}$

$(\sigma_{\textit{customer\_name = "Hayes"}}(\text{customer})))$

All customers' street and city

Hayes' street and city

customer-id	Customer-name	Customer-street	Customer-city	Hayes-street	Hayes-city
192-83-7465	Johnson	Alma St.	Palo Alto	Main St.	Harrison
019-28-3746	Smith	North St.	Rye	Main St.	Harrison
677-89-9011	Hayes	Main St.	Harrison	Main St.	Harrison
182-73-6091	Turner	Putnam Ave.	Stamford	Main St.	Harrison
321-12-3123	Jones	Main St.	Harrison	Main St.	Harrison
336-66-9999	Lindsay	Park Ave.	Pittsfield	Main St.	Harrison
019-28-3746	Smith	North St.	Rye	Main St.	Harrison

Rename Operation Example - 5

Step 4. Select only customer info. with the same street and city as Hayes

$\sigma_{\textit{customer\_street} = \textit{hayes\_street} \wedge \textit{customer\_city} = \textit{hayes\_city}}$

customer x ($\rho_{\textit{hayes\_addr(hayes\_street, hayes\_city)}}$ $\Pi_{\text{customer\_street, customer\_city}}$ ($\sigma_{\textit{customer\_name} = \text{“Hayes”}}$ (customer)))

customer-id	Customer-name	Customer-street	Customer-city	hayes_street	hayes_city
677-89-9011	Hayes	**Main St.**	**Harrison**	**Main St.**	**Harrison**
321-12-3123	Jones	**Main St.**	**Harrison**	**Main St.**	**Harrison**

Rename Operation Example - 6

Step 5. Filter out Hayes' record

σ *customer.customer_street = hayes_addr.street* $\wedge$ *customer.customer_city = hayes_addr.city* $\wedge$ ***customer.customer_name*** $\neq$ ***"****Hayes****"***

customer x (ρ *hayes_**addr(hayes_street, hayes_city)*** Πcustomer_street, customer_city (σ *customer_name = "Hayes"* (customer)))

customer-id	Customer-name	Customer-street	Customer-city	hayes_addr.street	hayes_addr.city
321-12-3123	Jones	Main St.	Harrison	Main St.	Harrison

Rename Operation Example - 6

Step 6. Retrieve only customer's name who live on the same street and in the same city as Hayes

$$\Pi_{customer.customer\_name}$$
$$(\sigma_{customer.customer\_street = hayes\_addr.street \wedge customer.customer\_city = hayes\_addr.city \wedge \textbf{customer.customer\_name} \neq \text{“Hayes”}}$$
$$(\text{customer} \times \rho_{hayes\_addr(street,\ city)} (\Pi_{customer\_street,\ customer\_city} (\sigma_{customer\_name = \text{“Hayes”}} (\text{customer})))))$$

Customer-name
Jones

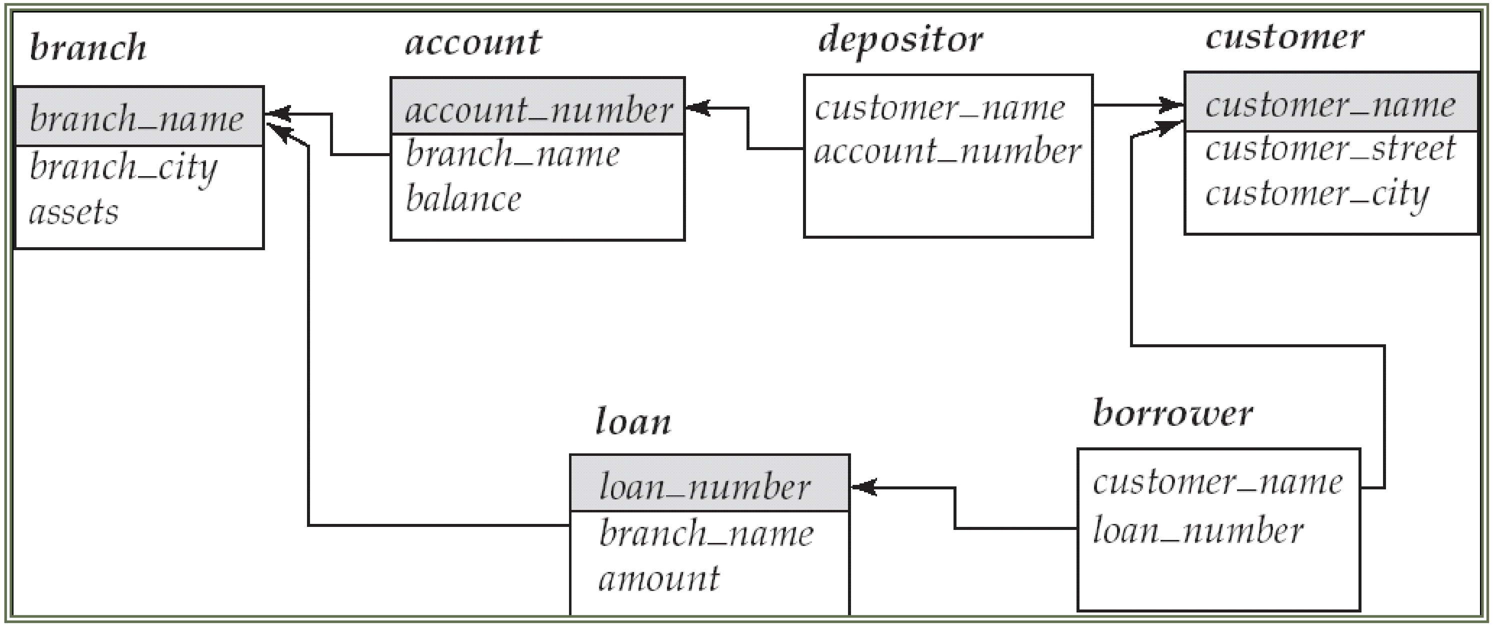
Formal Definition - 1

- **A basic expression in the relational algebra** consists of either one of the following:
  - **A relation variable**
  - **A constant relation**
    - **Written by listing tuples within { }**
    - e.g.) **{ (A-101, Downtown, 500), (A-102, Mianus, 700) }**
      - **a constant relation having two tuples, of length 3**
        - ***Of course, every tuple in a relation must of the same size since they all have the same number of attributes***

Formal Definition - 2

- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, 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

Banking Example



Refer to the handout provided

Example Queries - 1

- Find all loans of over $1200

- Find the loan number of each and every loan whose amount is greater than $1200

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

Example Queries - 2

- Find all loans of over $1200

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

- Find the loan number of each and every loan whose amount is greater than $1200

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

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

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

Example Queries - 3

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

Example Queries - 4

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

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

  - Query 2

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

  - Query 3

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

  - Query 4

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

Example Queries - 5

- 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.

Example Queries - 6

- 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.

$$\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{“Perryridge”}}$$

$$(\sigma_{borrower.loan\_number\ =\ loan.loan\_number}\ (\text{borrower x loan}))) \ -$$

$$\Pi_{customer\_name}(\text{depositor})$$

Example Queries - 7

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

  - The query is:

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

Classroom Exercise

- Find the names of all customers who have an account at Brighton branch
- Find all account numbers managed by any of branches in the city of Horseneck
- Find the names of customers who are living in “Ramkhamheng” street and have accounts in “Huamark” branch.
- Find all account numbers which have the same balance as A-222
- Find the smallest account balance
- Find the details of accounts (the account number, the branch in which the account is held and the current balance) that have the same balance in the same branch as **any** *of the accounts held by “Somchai”*

Additional Exercises

- Study the chapter 2 of the text thoroughly, especially 2.1 and 2.2
- Express the following queries using only "fundamental relational algebra operators" and show the resulting relation instances based on the banking example database schema and relation instances given in the handout:
  - Find all account number whose balance is greater than 500
  - Find all account numbers managed by any of branches in the city of Horseneck.
  - Find the names of all customers who have both a loan and an account
    - Hint: $A \cap B = A - (A - B)$
  - Find all account numbers which have the same balance as A-222
  - Find the name of all customers who have an account with the bank, along with his/her account number and the balance of the account.
  - Find the names of all customers who have an account at Brighton branch
  - Find the smallest account balance
  - Find the names of all customer who have accounts in any of branches whose assets is below 2000000
  - Find the names of customers who are living in "Ramkhamheng" street and have accounts in "Huamark" branch.
  - Find all the customers who have at least one account and one loan in a same branch
  - Find the account numbers and names of the account holders whose balance is equal to balance of any of accounts held by "Somchai" and held in the same branch as to having the Somchai's account having the same balance

ToDo Task

- Complete Worksheet 1

  Due: Midnight of the next Mon

  Submission: submit 1 pdf file per group via MS Team