Introduction to Databases
《数据库引论》

Lecture 2: Relational Model and Relational Algebra
第2讲：关系模型与关系代数

周水庚 / Shuigeng Zhou

邮件: sgzhou@fudan.edu.cn 网址：admis.fudan.edu.cn/sgzhou

复旦大学计算机科学技术学院

Content of the Course

- Part 0: Overview
  - Lect. 0/1 (Feb. 20) - Ch1: Introduction
- Part 1 Relational Databases
  - Lect. 2 (Feb. 27) - Ch2: Relational model (data model, relational algebra)
  - Lect. 3 (Mar. 6) - Ch3&4: SQL (Introduction and intermediate)
  - Lect. 4 (Mar. 13) - Ch5: Advanced SQL
- Part 2 Database Design
  - Lect. 5 (Mar. 20) - Ch6: Database design based on E-R model
  - Lect. 6 (Mar. 27) - Ch7: Relational database design (Part I)
  - Lect. 7 (Apr. 3) - Ch7: Relational database design (Part II)
- Midterm exam: Apr. 10
- Part 3 Data Storage & Indexing
  - Lect. 7 (Apr. 17) - Ch12/13: Storage systems & structures
  - Lect. 8 (Apr. 24) - Ch14: Indexing
- Part 4 Query Processing & Optimization
  - May 1, holiday, no classes
  - Lect. 9 (May 8) - Ch15: Query processing
  - Lect. 10 (May 15) - Ch16: Query optimization
- Part 5 Transaction Management
  - Lect. 11 (May 22) - Ch17: Transactions
  - Lect. 12 (May 29) - Ch18: Concurrency control
  - Lect. 13 (Jun. 5) - Ch19: Recovery system
  - Lect. 14 (Jun. 5) – Course review

Final exam: 13:00-15:00, Jun. 18

Two Tables of the University Database

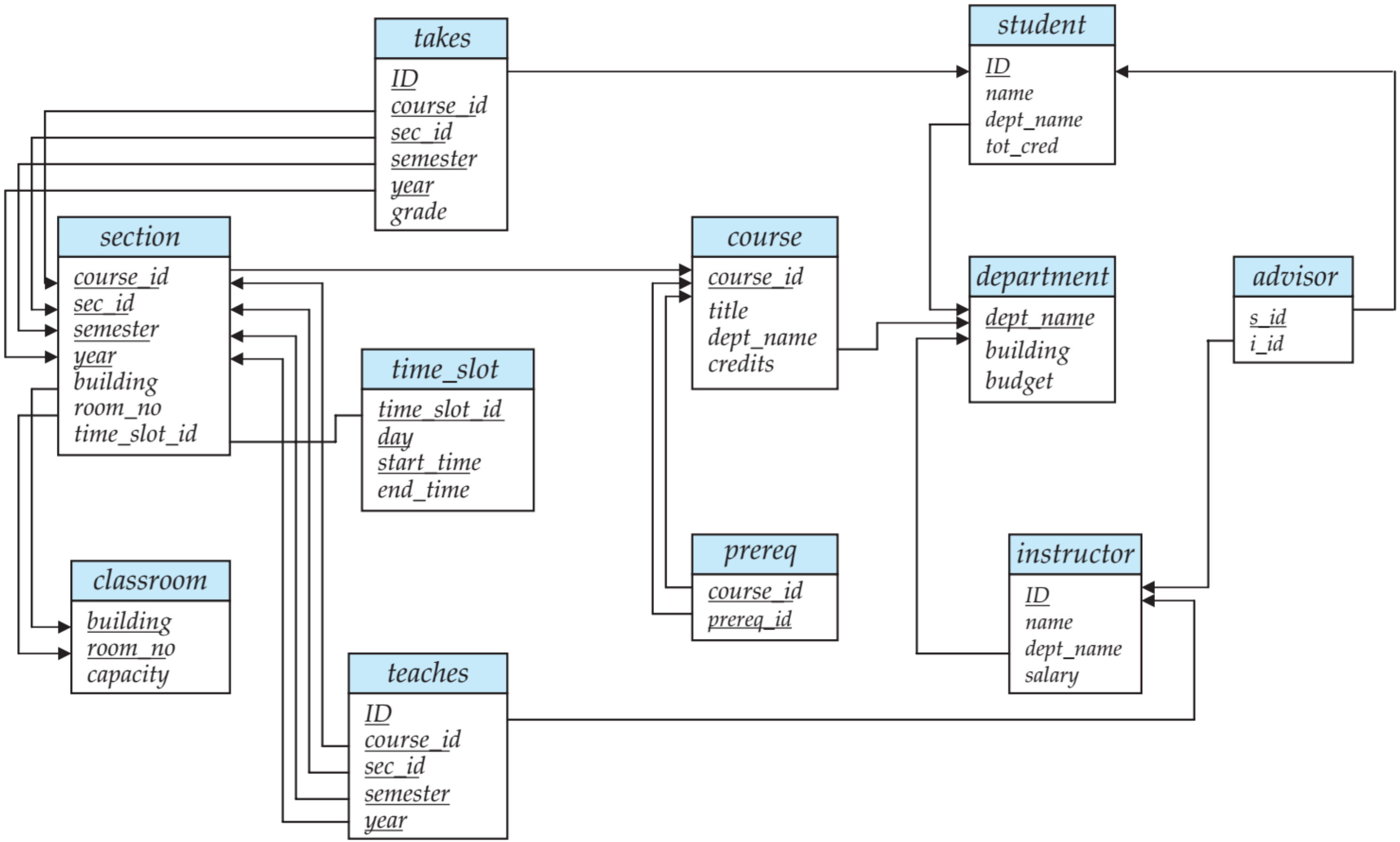
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor table

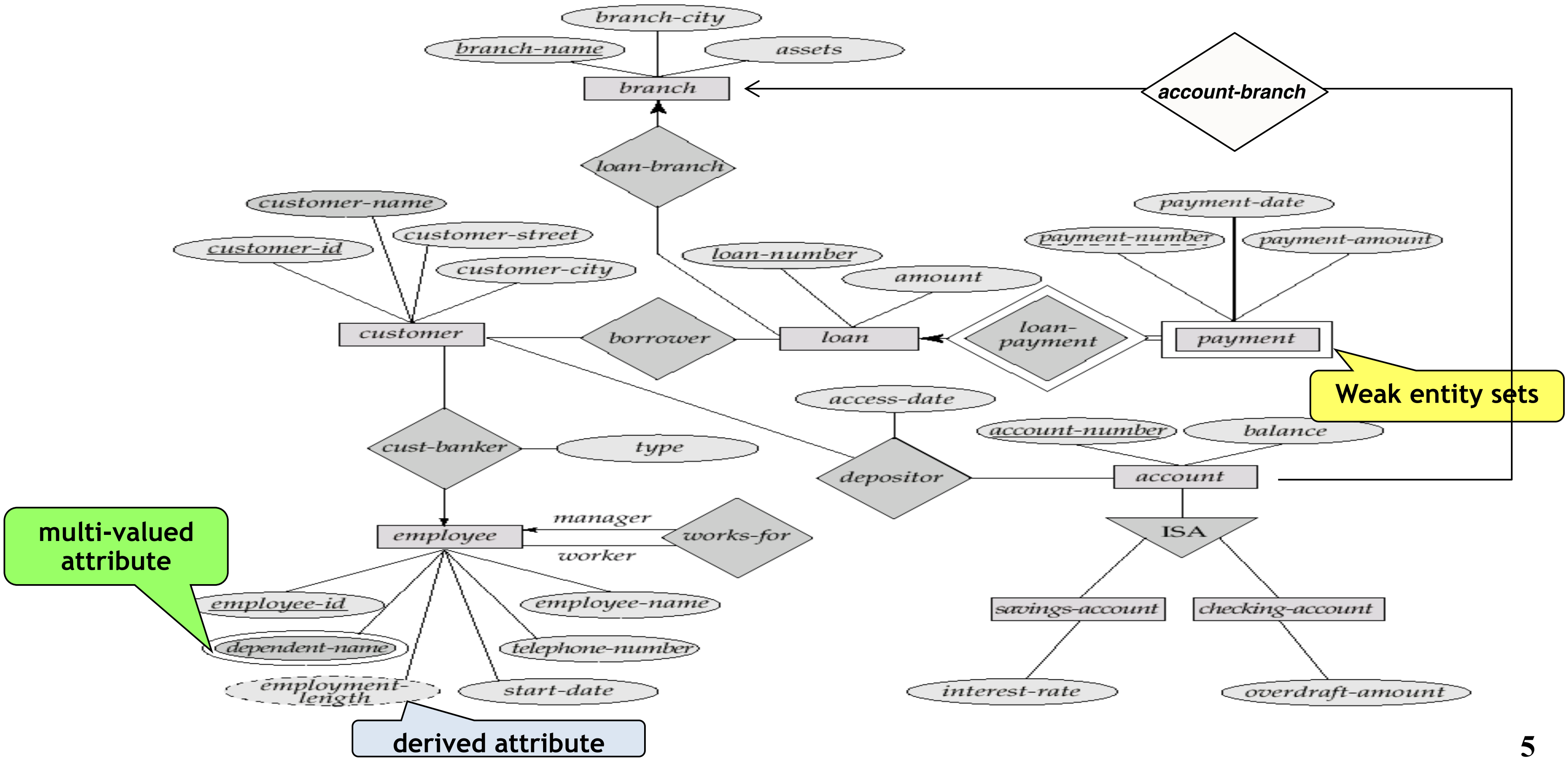
ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Student table

Schema Diagram of the University Database



E-R Diagram for a Banking Enterprise



The Banking Database Schema

- *branch* = (*branch_name*, *branch_city*, *assets*)
- *customer* = (*customer_id*, *customer_name*, *customer_street*, *customer_city*)
- *loan* = (*loan_number*, *amount*)
- *account* = (*account_number*, *balance*)
- *employee* = (*employee_id*, *employee_name*, *telephone_number*, *start_date*)

- *dependent_name* = (*employee_id, dname*) (derived from a multivalued attribute)

- *account_branch* = (*account_number*, *branch_name*)
- *loan_branch* = (*loan_number*, *branch_name*)
- *borrower* = (*customer_id, loan_number*)
- *depositor* = (*customer_id, account_number*, *access_date*)
- *cust_banker* = (*customer_id*, *employee_id*, *type*)
- *works_for* = (*worker_employee_id*, *manager_employee_id*)

- *payment* =(*loan_number*,*payment_number*,*payment_date*,*payment_amount*)

- *savings_account* = (*account_number*, *interest_rate*)
- *checking_account* = (*account_number*, *overdraft_amount*)

Outline

- ☞ **Relational Database Model**
  - The structure of a relation
  - Relational database and
  - Keys
  - Database schema
- Relational Algebra
  - Relational query languages
  - Relational operations

An Example of Relation/Table

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Basic Structure of a Relation

- Given sets $D_1, D_2, \cdots, D_n$, a relation r is a subset of $D_1 \times D_2 \times \cdots \times D_n$, i.e., a set of n-tuples $(a_1, a_2, \cdots, a_n)$ where each $a_i \in D_i (i = 1, \ldots n)$

n-tuples: n元组

- E.g., 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

Attribute (属性)

- Each relation consists of a set of **attributes** $A_1, A_2, \ldots, A_n$
- The **domain** of an attribute is the whole set of available and legal values of the attribute
- Attribute values are (normally) required to be **atomic (原子性)**
  - Multi-valued attributes and composite attributes are not atomic
    - 多值属性：电话号码；复合属性：通信地址
- The special value **null** is a member of every domain. It may cause complications in the definition of many operations

Relation Schema (关系模式)

- $A_1, A_2, \ldots, A_n$ are attributes, and $R = (A_1, A_2, \ldots, A_n)$ is a relation schema,
- e.g.,
  - *instructor_schema =(id, name, dept_name, salary)*
  - *customer_schema=(custom_id, custom_ name, custom_ street, custom_ city)*

- $r(R)$ is a relation on the relation schema R,
- e.g.,
  - *instructor(instructor_schema)*
  - *customer(customer_schema)*

Relation Instance (关系实例)

- **A relation instance** corresponds to the current values of a relation, which is specified by a **table**
- An element t of r is a **tuple** (元组), represented by a **row** in the table

Attributes/Columns

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Tuples/Rows

Relation vs. Variable

- Relation schema vs. Variable type
- Relation instance vs. Variable value
- For example
  - int vs. *customer_schema* =(id, *name*, *street*, *city*)
  - int *A* vs. customer(customer_schema)
  - *A*=10 vs.

Jones Smith Curry Lindsay	Main North North Park	Harrison Rye Rye Pittsfield

Relations are Unordered

- The order of tuples/attributes in a relation is irrelevant. Tuples could be stored in an arbitrary order
- E.g., instructor relation with unordered tuples

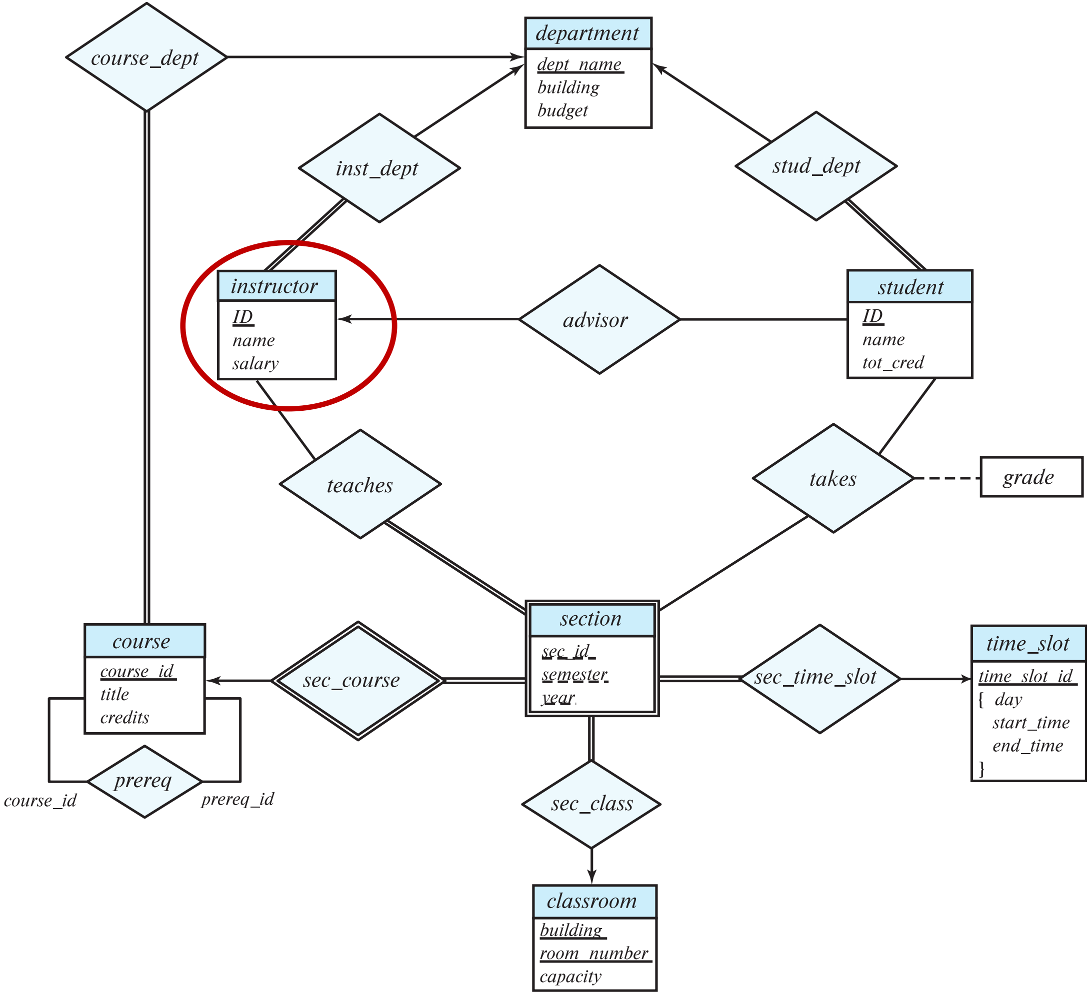
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Outline

- ☞ **Relational Database Model**
  - The structure of a relation
  - Relational database
  - Keys
  - Database schema
- Relational Algebra
  - Relational query languages
  - Relational operations

Database

- A database consists of multiple relations
- Why NOT use a single relation?
- Storing all information as a single relation results in
  - repetition of information, e.g., one department has many students, record the information of both department and student
  - the need for null values, e.g., represent a customer without an account
- How many relations should have?
  - Normalization (规范化) theory (Chapter 7) deals with how to design relational schemas



E-R Diagram for University Database

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

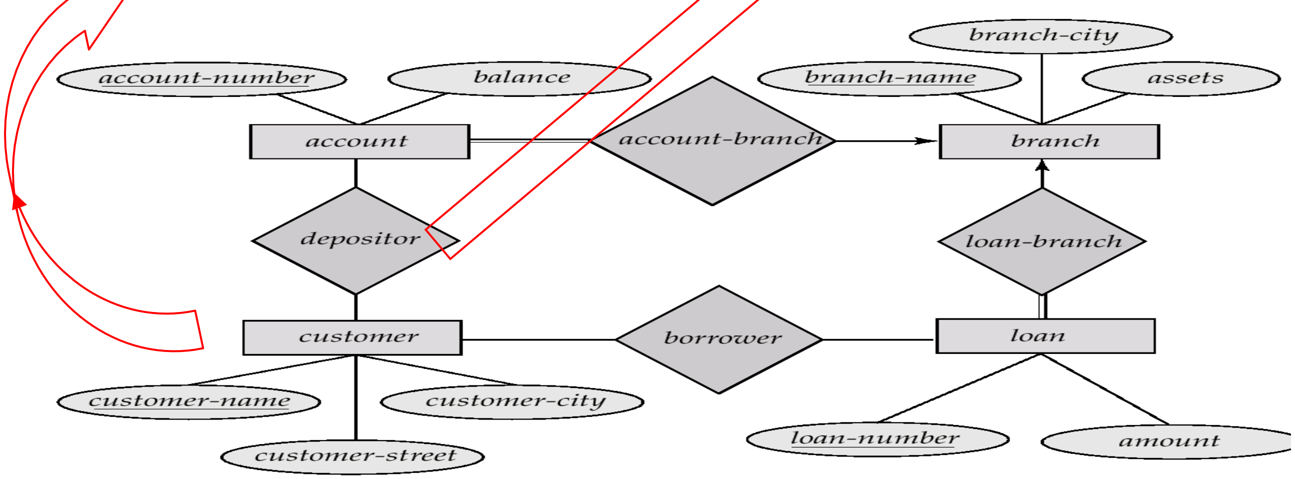
Instructor relation

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 customer 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

The depositor Relation



E-R Diagram for the Banking Enterprise

Outline

- ☞ Relational Database Model
  - The structure of a relation
  - Relational database
  - Keys
  - Database schema
- Relational Algebra
  - Relational query languages
  - Relational operations

Keys (码、键)

- **Superkey（超码）**
  - Let $K \subseteq R$, K is a superkey of relation schema R if the values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
  - E.g., {*instructor_id*}, {*instructor_id, instructor_name*} and {*instructor_name*} are superkeys of *instructor*, if no two instructors have the same name
  - If tuples $t_1 \neq t_2$, then $t_1[K] \neq t_2[K]$
- **Candidate key（候选码）**
  - K is a candidate key if K is minimal
  - E.g., {*instructor_name*} is a candidate key for *instructor*, since it is a superkey (assuming no two instructors have the same name)
- **Primary key（主码）/ Primary key constraint**
  - A candidate key is chosen by the DB designer to identify tuples within a relation

Keys (Cont.)

- Foreign key(外键/外码)
  - A relation schema $\boldsymbol{R_1}$, may include among its attributes the primary key of another relation schema $\boldsymbol{R_2}$. This attribute is called a **foreign key** from $\boldsymbol{R_1}$, referencing $\boldsymbol{R_2}$
  - The relation $\boldsymbol{r_1}$ is called the **referencing relation** (参照关系) of the **foreign key dependency**, and $\boldsymbol{r_2}$ is called the **referenced relation** (被参照关系) of the foreign key dependency
- Foreign key constraint / Referential integrity constraint (外键约束/参照完整性约束)
  - The values appearing in specified attributes of any tuple in the referencing relation should also appear in specified attributes of at least one tuple in the referenced relation

The University Database Schema

- *classroom(building, room_number, capacity)*
- *department(dept_name, building, budget)*
- *course(course_id, title, dept_name, credits)*
- *instructor(ID, name, dept_name, salary)*
- *section(course_id, sec_id, semester, year, building, room_number, time_slot_id)*
- *teaches(ID, course_id, sec_id, semester, year)*
- *student(ID, name, dept_name, tot_cred)*
- *takes(ID, course_id, sec_id, semester, year, grade)*
- *advisor(s_ID, i_ID)*
- *time slot(time_slot_id, day, start_time, end_time)*
- *prereq(course_id, prereq_id)*

The Banking Database Schema

- *branch* = (*branch_name*, *branch_city*, *assets*)
- *customer* = (*customer_id*, *customer_name*, *customer_street*, *customer_city*)
- *loan* = (*loan_number*, *amount*)
- *account* = (*account_number*, *balance*)
- *employee* = (*employee_id*, *employee_name*, *telephone_number*, *start_date*)

- *dependent_name* = (*employee_id, dname*) (derived from a multivalued attribute)

- *account_branch* = (*account_number*, *branch_name*)
- *loan_branch* = (*loan_number*, *branch_name*)
- *borrower* = (*customer_id, loan_number*)
- *depositor* = (*customer_id, account_number*, *access_date*)
- *cust_banker* = (*customer_id*, *employee_id*, *type*)
- *works_for* = (*worker_employee_id*, *manager_employee_id*)

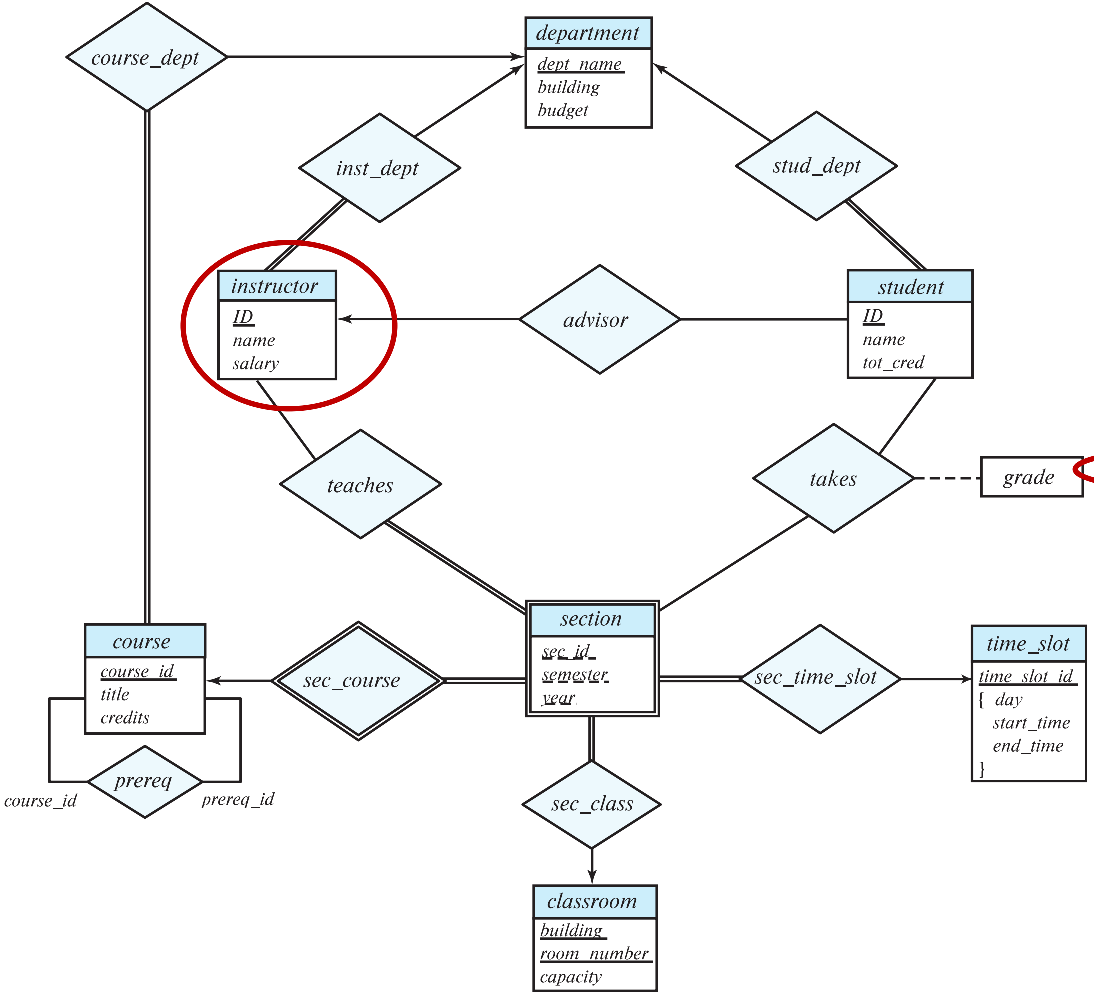
- *payment* =(*loan_number*,*payment_number*,*payment_date*,*payment_amount*)

- *savings_account* = (*account_number*, *interest_rate*)
- *checking_account* = (*account_number*, *overdraft_amount*)

Determining Keys from E-R Sets

- Strong entity set: has a primary key
- Weak entity set: may not have sufficient attributes to form a primary key
  - Discriminator (分辨符) plus the Key of the identifying entity set (标识实体集, or owner entity set 属主实体集)
- Relationship set
  - Union of keys of the related entity sets

(discussed later in Chapter 6)



classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

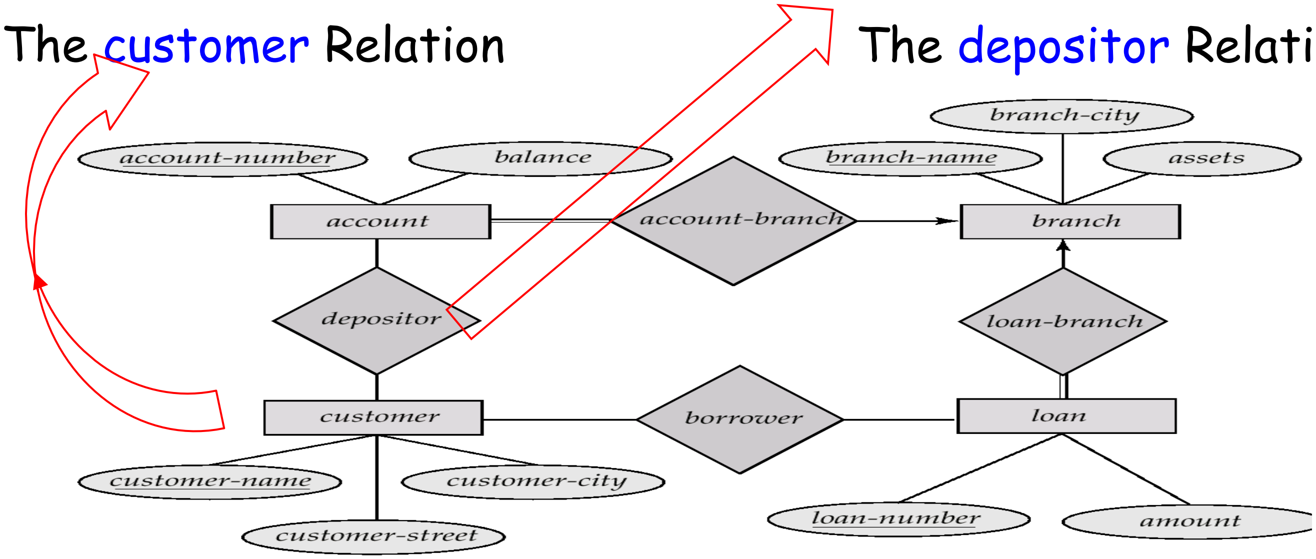
E-R Diagram for University Database

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 customer 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

The depositor Relation



E-R Diagram for the Banking Enterprise

Outline

- ☞ Relational Database Model
  - The structure of a relation
  - Relational database
  - Keys
  - Database schema
- Relational Algebra
  - Relational query languages
  - Relational operations

Database Schema

- **Database schema**
  - All the schemas of relations, along with primary key and foreign key dependencies in a database consist of the database's schema

- **Database schema diagram (模式图)**
  - A database schema can be depicted pictorially by a schema diagram

Schema Diagram

branch (*branch_name*, *branch_city*, *assets*)

customer (*customer_name*, *customer_street*, *customer_city*)

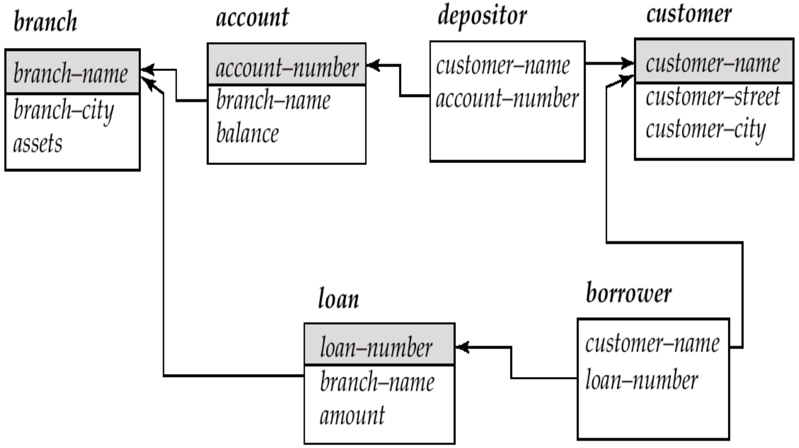
account (*account_number*, *branch_name*, *balance*)

loan (*loan_number*, *branch_name*, *amount*)

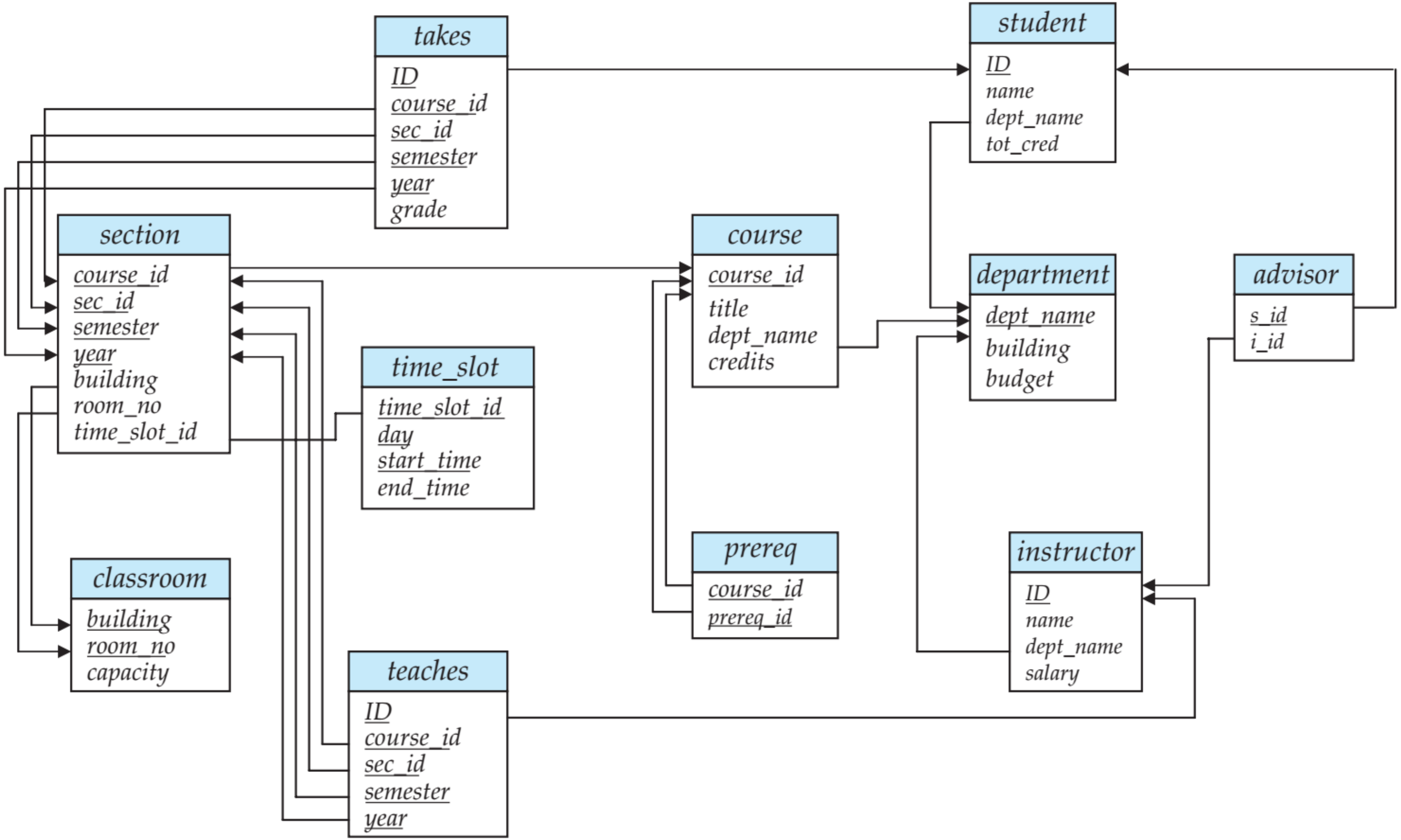
depositor (*customer_name*, *account_number*)

borrower (*customer_name*, *loan_number*)

Banking database



Schema Diagram (模式图)
for the banking enterprise



Schema diagram for the university database

Outline

- **Relational Database Model**
  - The structure of a relation
  - Relational database
  - Keys
  - Database schema
- ☞ **Relational Algebra**
  - Relational query languages
  - Relational operations

Relational Query Languages

- Query Languages used to request information from the database
  - Imperative languages, functional languages, declarative languages
- Categories of languages
  - Procedural
    - Relational Algebra（关系代数）：functional language
  - Non-procedural
    - SQL（结构化查询语言）：mainly, it is declarative, but it also has imperative, functional features
    - Tuple Relational Calculus（元组关系演算）
      - $R-S = \{t \mid R(t) \wedge \neg S(t)\}$, $R \cup S = \{t \mid R(t) \vee S(t)\}$
    - Domain Relational Calculus（域关系演算）
      - $\{<A, B, C> \mid <A, B, C> \in Student \wedge C = \text{"Monitor"}\}$

Outline

- **Relational Database Model**
  - The structure of a relation
  - Relational database
  - Keys
  - Database schema
- ☞ **Relational Algebra**
  - Relational query languages
  - Relational operations

Relational Algebra

- A procedural language consisting of a set of operations that take one or more relations as input and produce a new relation as the result
- Six basic operations
  - Select (选择)；水平选择，选择行/元组
  - Project (投影)；垂直选择，选择列/属性
  - Union (集合并)
  - Set difference (集合差)
  - Cartesian product (笛卡尔积)
  - Rename (重命名)
- These operators take one or two relations as inputs and give a new relation as a result

Select Operation

- Notation: $\sigma_P(r) = \{t | t \in r \ and \ P(t)\}$
  - P is the selection predicate(选择谓词) consisting of ∧(and), ∨(or), ¬(not), =, ≠, <, >, ≤, ≥
  - E.g.,

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

relation r

A	B	C	D
α	α	1	7
β	β	23	10

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

Select Operation (Cont.)

- E.g., select those tuples of the instructor relation where the instructor is in the "Physics" department
  - Query

$$\sigma_{\text{dept\_name= ``Physics''}}(\textit{instructor})$$

  - Result

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor relation

Select Operation (Cont.)

- Allow comparisons using **=, ≠, >, ≥, <, ≤** in the selection predicate.
- Can combine several predicates into a larger predicate by using the connectives（连接词）: **∧ (and), ∨ (or), ¬ (not)**
  - **Example:** Find the instructors in Physics with a salary greater $90,000

$$\sigma_{dept\_name = \text{"Physics"} \wedge salary > 90{,}000}(instructor)$$

- The select predicate may include comparisons between two attributes.
  - **Example:** find all departments whose name is the same as their building name:

$$\sigma_{dept\_name = building}(department)$$

Project Operation

- Notation: $\Pi_{A_1,A_2,\ldots,A_k}(r)$
  - $A_1, A_2, \ldots, A_k$ 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 are removed from result, since relations are sets
  - E.g.,

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 (Cont.)

- E.g., eliminate the *dept_name* attribute of *instructor*
- Query:

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

- Result:

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor relation

Union Operation

- Notation: $r \cup s = \{t | t \in r \text{ or } t \in s\}$
  - r, s must have the same arity (同元的), i.e., the same number of attributes
  - The attribute domains must be compatible (相容的)
    - E.g., the 2nd column of r deals with the same type of values as does the 2nd column of s
  - E.g., find all courses taught in the Fall 2022 semester, or in the Spring 2023 semester, or in both:

$$\Pi_{course\_id}(\sigma_{semester=\text{"Fall"} \wedge year=2022}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{"Spring"} \wedge year=2023}(section))$$

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

relations r, s

A	B
α	1
α	2
β	1
β	3

$r \cup s$

Set Difference Operation

- **Notation:** $r - s = \{t | t \in r \text{ and } t \notin s\}$
  - Set differences must be taken between compatible relations, i.e., r and s must have the same arity and attribute domains
  - E.g.,

r:

A	B
α	1
α	2
β	1

s:

A	B
α	2
β	3

relations r, s

A	B
α	1
β	1

$r - s$

Set Difference Operation (Cont.)

- E.g., to find all courses taught in the Fall 2022 semester, but not in the Spring 2023 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{"Fall"} \wedge year=2022}(section)) -$$
$$\Pi_{course\_id}(\sigma_{semester=\text{"Spring"} \wedge year=2023}(section))$$

Cartesian Product Operation

- **Notation:** $r \times s = \{tq | t \in r \; and \; q \in s\}$
  - The attributes of $r(R)$ and $s(S)$ should be disjoint, i.e., $R \cap S = \emptyset$
  - If the attributes of $r(R)$ and $s(S)$ are not disjoint, then renaming must be used

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

relations r, 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

$r \times s$

instructor x teaches table

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Cartesian Product Operation (Cont.)

- $\sigma_{instructor.id\ =\ teaches.id}(instructor \times teaches))$

instructor.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

Composition of Operations

- Build expressions using multiple operations
  - E.g., $\sigma_{A=C}(r \times s)$

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

relations r, 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

$r \times s$

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

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

Rename Operation (更名运算)

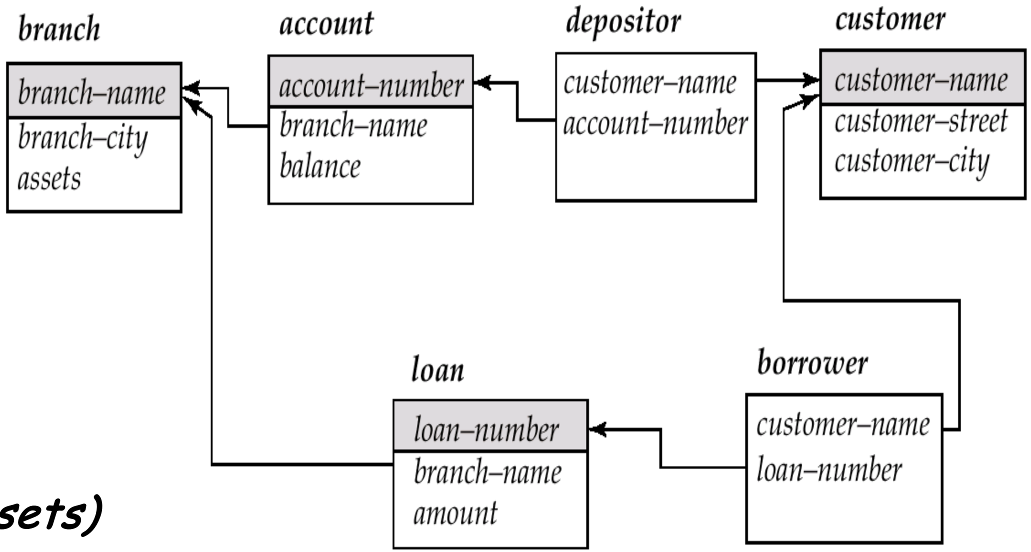
- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
  - E.g., $\rho_X(E)$ returns the expression E under the name X

- If a relational-algebra expression E has arity n
  - $\rho_{X(A_1,A_2,\ldots,A_n)}(E)$ returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots, A_n$

Notes about Relational Languages

- Each query input is a table (or a set of tables)
- Each query output is a table.
- All data in the output table appears at least in one of the input tables

Schema for Following Examples



branch (<u>branch_name</u>, branch_city, assets)

customer (<u>customer_name</u>, customer_street, customer_city)

account (<u>account_number</u>, branch_name, balance)

loan (<u>loan_number</u>, branch_name, amount)

depositor (<u>customer_name, account_number</u>)

borrower (<u>customer_name, loan_number</u>)

Example Queries (1)

- Find all loans of over $1200

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

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

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

Example Queries (2)

- 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

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

- 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

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

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

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

Example Queries (5)

- 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

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

Relational Expressions

- A basic expression in the relational algebra consists of either of the following
  - A relation in the database
  - A constant relation, e.g., {(22222, Einsteir, Physics, 9500), (76543, Singh, Finance, 80000)}
- The result of any relational operation on a basic expression is relational-algebra expression
- 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
  - $\Pi_s(E_1)$, 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

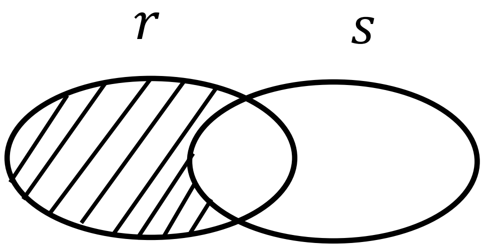
Additional Operations

- Additional operations
  - Set intersection (集合交)
  - Natural join (自然连接)
  - Outer join （外连接)
  - Division (除)
  - Assignment (赋值)

- Additional operations do not add any power to the relational algebra, but simplify common queries

Set Intersection Operation

- **Notation:** $r \cap s = \{t | t \in r \text{ and } t \in s\}$
  - r, s have the same arity
  - the attributes of r and s are compatible
  - Note: $r \cap s = r - (r - s)$



$r - s$

Relations r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

$r \cap s$:

A	B
α	2

Set Intersection Operation (Cont.)

- **E.g.**, Find the set of all courses taught in both the Fall 2022 and the Spring 2023 semesters.

$\Pi_{course\_id} (\sigma_{semester= \text{"Fall"} \land year=2022}(section)) \cap$

$\Pi_{course\_id} (\sigma_{semester= \text{"Spring"} \land year=2023}(section))$

Natural Join Operation

- Notation: $r \bowtie s$
- Let r and s be the relations on schemas R and S respectively. Then $r \bowtie s$ 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

- E.g., $R = (A,\ B,\ C,\ D)$, $S = (E, B, D)$
  - Result schema: (A, B, C, D, E)
  - $r \bowtie s$ is defined as: $\Pi_{r.A,\, r.B,\, r.C,\, r.D,\, s.E}(\sigma_{r.B=s.B\ \wedge r.D=s.D}(r \times s))$

Natural Join Operation – Example

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	∈

s

$r \bowtie s$:

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(cont.)

- Let $r(R)$ and $s(S)$ be relations without any attributes in common, i.e., $R \cap S = \emptyset$. Then, $r \bowtie s = r \times s$
- θ-join operation
  - An extension to the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation.
  - Consider relations $r(R)$ and $s(S)$, and let θ be a predicate on attributes in the schema $R \cup S$. The theta join $r \bowtie_\theta s$ is defined as follows: $r \bowtie_\theta s = \sigma_\theta(r \times s)$

Join Operation – Example

Relations r, s:

r:

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

s:

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2

$r \bowtie s$

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

$r \bowtie_{r.B=s.B} s$

A	R.B	C	S.B	E
a1	b1	5	b2	7
a1	b1	5	b3	10
a1	b2	6	b2	7
a1	b2	6	b3	10
a2	b3	8	b3	10

$r \bowtie_{C<E} s$

Outer Join

- An extension of the join operation that avoids loss of information
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join
- Uses **null** values:
  - null signifies that the value is unknown or does not exist
  - All comparisons involving null are (roughly speaking) false by definition.

Outer Join – Example

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation *loan*

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

Relation *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Inner 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*

Left 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

Right 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

Full Outer Join: *loan* ⟗ *borrower*

Division Operation

- Notation: $r \div s$
  - r and s are relations on schemas R and S, respectively
    - $R = (A_1, \dots, A_m, B_1, \dots, B_n)$
    - $S = (B_1, \dots, B_n)$
  - The result of $r \div s$ is a relation on schema $R - S = (A_1, \dots, A_m)$, i.e.,

$$r \div s = \{t | t \in \Pi_{R-S}(r) \land \forall u \in s(tu \in r)\}$$

- A tuple t is in $r \div s$ if and only if both of two conditions hold:
  - t is in $\Pi_{R-S}(r)$
  - For every tuple t_s in s, there is a tuple t_r in r satisfying:
    - $t_r[S] = t_s[S]$
    - $t_r[R - S] = t$

Division Operation – Example

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 (Cont.)

- Definition in terms of the basic algebra operation
  - Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$
  - $r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$
- To see why
  - $\Pi_{R-S,S}(r)$ simply reorders attributes of r
  - $\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$

Assignment Operation

- The assignment operation (←) provides a convenient way to express complex queries
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query
  - Assignment must be made to a temporary relation variable
- Example: write $r \div s$ as:
  - $temp_1 \leftarrow \Pi_{R-S}(r)$
  - $temp_2 \leftarrow \Pi_{R-S}((temp_1 \times s) - \Pi_{R-S,S}(r))$
  - $result = temp_1 - temp_2$
- The result to the right of the ← is assigned to the relation variable on the left of the ←

Example Queries (6)

- Find all the customers who have accounts from at least the "Downtown" and the "Uptown" branches
- Query 1
  - $\Pi_{CN}(\sigma_{BN="Downtown"}(depositor \bowtie account)) \cap \Pi_{CN}(\sigma_{BN="Uptown"}(depositor \bowtie account))$
  - where CN denotes customer_name and BN denotes branch_name
- Query 2
  - $\Pi_{customer\_name, branch\_name}(depositor \bowtie account) \div \rho_{temp(branch\_name)}(\{(\text{"Downtown"}), (\text{"Uptown"})\})$
  - Note that Query2 uses a constant relation

Example Queries (7)

- Find all customers who have an account at all branches located in Shanghai
  - $\Pi_{customer\_name, branch\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_city="Shanghai"}(branch))$

Extended Relational Algebra Operations

- Generalized Projection（广义投影）
- Aggregate Functions （聚合函数）

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list $\Pi_{F_1, F_2, \ldots, F_n}(E)$
  - E is any relational-algebra expression
  - Each of $F_1, F_2, \ldots, F_n$ is a arithmetic expression involving constants and attributes in the schema of E
- Given relation *credit_info(customer_name, limit, credit_balance)*, find how much more each person can spend:
  - $\Pi_{customer\_name,\ limit - credit\_balance}(credit\_info)$

Aggregate Functions and Operations

- **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
  - ${}_{G_1,G_1,\ldots,G_n}\mathcal{G}_{F_1(A_1),F_2(A_2),\ldots,F_n(A_n)}(E)$ (先分组,再聚合)
    - E is any relational-algebra expression
    - $G_1, G_2, \ldots, G_n$ is a list of attributes on which to group (can be empty)
    - Each F_i is an aggregate function （再做聚合）
    - Each A_i is an attribute name

Aggregate Operation – Example

Relation *r*:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{sum(c)}$ (r) :

Sum(C)
27

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 $\mathcal{G}$ *sum(balance)* (*account*)

branch_name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$_{branch\_name}\mathcal{G}_{\ sum(balance)\ as\ sum\_balance}(account)$$

Null Values

- It is possible for tuples to have a null value 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
  - Is an arbitrary decision? Could have returned null as result instead?
  - We follow the semantics of SQL in its handling of null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow SQL

Null Values

- Three-valued logic (三值逻辑) 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
  - In SQL "P is unknown" evaluates to true if predicate P evaluates to unknown
- Result of select predicate is treated as false if it evaluates to unknown

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operation

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the **selected tuples** are removed from the database
  - Can only delete whole tuples
  - cannot delete values on particular attributes
- A **deletion** is expressed in relational algebra by:
  - $r \leftarrow r - E$, where r is a relation and E is a relational algebra query

Deletion Examples

- Delete all account records in the Perryridge branch.

  $account \leftarrow account - \sigma_{branch\_name = \text{"Perryridge"}}(account)$

- Delete all loan records with amount in the range of 0 to 50

  $loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$

- Delete all accounts at branches located in Shanghai

  $r_1 \leftarrow \sigma_{branch\_city = \text{"Shanghai"}}(account \bowtie branch)$

  $r_2 \leftarrow \Pi_{account\_number,\ branch\_name,\ balance}(r_1)$

  $r_3 \leftarrow \Pi_{customer\_name,\ account\_number}(r_2 \bowtie depositor)$

  $account \leftarrow account - r_2$

  $depositor \leftarrow depositor - r_3$

Insertion

- To **insert** data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- In relational algebra, an **insertion** is expressed by:
  - $\boldsymbol{r} \leftarrow \boldsymbol{r} \cup \boldsymbol{E}$, where $\boldsymbol{r}$ is a relation and $\boldsymbol{E}$ is a relational algebra expression.
  - The insertion of a single tuple is expressed by letting $\boldsymbol{E}$ be a constant relation containing one tuple

Insertion Examples

- Insert information in the database specifying that Smith has \$1200 in account A_973 at the Perryridge branch.

$$account \leftarrow account \cup \{(A\_973, \text{“Perryridge”}, 1200)\}$$
$$depositor \leftarrow depositor \cup \{(\text{“Smith”}, A\_973)\}$$

- Provide as a gift for all loan customers in the Perryridge branch, a \$200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch\_name = \text{“Perryridge”}}(borrower \bowtie loan))$$
$$account \leftarrow account \cup \Pi_{loan\_number,\ branch\_name,\ 200}(r_1)$$
$$depositor \leftarrow depositor \cup \Pi_{customer\_name,\ loan\_number}(r_1)$$

Updating

- A mechanism to change a value in a tuple without changing all other values in the tuple
- Use the **generalized projection** operator to do this task
  - $r \leftarrow \Pi_{F_1, F_2, \ldots, F_i}(r)$
  - Each F_i is either
    - the ith attribute of r, if the ith attribute is not updated, or
    - if the attribute to be updated, F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

Update Examples

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.05}\ (account)$$

where AN, BN and BAL stand for *account_number*, *branch_name* and *balance*, respectively

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.06}\ (\sigma_{BAL\ >\ 10000}\ (account)) \cup \Pi_{AN,\ BN,\ BAL\ *1.05}\ (\sigma_{BAL\ \leq\ 10000}\ (account))$$

Updating

- To select some tuples from r to update, we can use the following expression:

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(\sigma_P(r)) \cup (r - \sigma_P(r))$$

where P denotes the selection condition that chooses which tuples to update

Views（视图）

- In some cases, it is not desirable for all users to see the entire logical model
- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by

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

- Any relation that is not of the conceptual model but is made visible to a user as a "**virtual relation**" is called a **view**

View Definition

- A view is defined using the create view statement which has the form

  create view v as < query expression >

- Once a view is defined, the **view name** can be used to refer to the **virtual relation** that the view generates
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, **a view definition** causes **the saving of an expression**
  - the expression is substituted into queries using the view

View Examples

- Consider the view (named *all_customer*) consisting of branches and their customers

create view *all_customer* **as**

$\Pi_{branch\_name,\ customer\_name}\ (depositor \bowtie account)$

$\cup\ \Pi_{branch\_name,\ customer\_name}\ (borrower \bowtie loan)$

- We can find all customers of the Perryridge branch by writing:

$\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{"Perryridge"}}\ (all\_customer))$

Updates Through View

- **Must be translated to modifications of the actual relations**
- Consider the person who needs to see all loan data in the loan relation except amount. The view given to the person, *branch_loan*, is defined as:

  create view branch_loan as $\Pi_{branch\_name, loan\_number}(loan)$

- Since we allow a view name to appear wherever a relation name is allowed, the person may write:

  $branch\_loan \leftarrow branch\_loan \cup \{(\text{"Perryridge"}, L\_37)\}$

- An insertion into relation *loan* requires a value for amount. The insertion can be handled by either.
  - rejecting the insertion
  - inserting a tuple (L_37, “Perryridge”, null)

Updates Through Views (Cont.)

- Some updates through views are impossible to translate into database relation updates

$$\textbf{create view } v \textbf{ as } (\sigma_{branch\_name="Perryridge"}(account))$$

$$v \leftarrow v \cup (L\_99, \text{"Downtown"}, 23)$$

- Others cannot be translated uniquely

create view *all_customer* **as**

$$\Pi_{branch\_name,\ customer\_name}(depositor \bowtie account)$$

$$\cup\ \Pi_{branch\_name,\ customer\_name}(borrower \bowtie loan)$$

$$all\_customer \leftarrow all\_customer \cup \{(\text{Perryridge}, \text{"}John\text{"})\}$$

  - Have to choose loan or account, and create a new loan/account number

Views Defined Using Other Views

- One view may be used to define another view
- A view relation v_1 is said to depend directly (直接依赖) on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to depend（依赖）on view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be recursive (递归) if it depends on itself

View Expansion

- A way to define the meaning of views defined in terms of other views
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations
- View expansion of an expression repeats the following replacement step:

  repeat

      Find any view relation v_i *in* e_1

      Replace the view relation v_i *by the expression defining* v_i

  until no more view relations are present in e_1
- As long as the view definitions are not recursive, this loop will terminate

Summary

- Relation/table
  - Attributes, domain, null value
  - Keys: superkeys, candidate keys, primary keys, foreign keys
  - Relational schema, relation instance, tuple
- Relational Database
  - A set of relations connected by foreign-key constratints
  - Database schema/database schema diagram
- Relational algebra
  - Basic operations
    - select σ, project Π, Cartesian product $\times$, set union $\cup$, set difference -, rename ρ
  - Additional operations
    - set intersection $\cap$, natural join $\bowtie$, conditional join, outer join, division $\div$, assignment $\leftarrow$
  - Generalized projection and aggregate functions
  - Insertion, delete, update
- View

Homework

- Exercises
  - 2.1, 2.6, 2.7, 2.8, 2.15, 2.18
- Submission
  - E-learning系统，上传单个word或者PDF文件
  - Deadline: 12:00pm, March 5, 2025

End of Lecture 2