UNIT 3 – RELATIONAL MODEL

Topics

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations

Introduction To Relational Model

- **Relational Database Model** is the most common model in industry today.
- A relational database is based on the relational model developed by E.F. Codd.
- The relational model- collection of tables and the relationships among those data.

Properties of a relation

- Each relation contains only one record type.
- Each relation has a fixed number of columns that are explicitly named. Each attribute name within a relation is unique.
- No two rows(tuples) in a relation are the same.
- Each item or element in the relation is atomic.
- Rows have no ordering associated with them.
- Columns have no ordering associated with them.

Relational Terminology

Terms	Definition
Relation	Set of rows(tuples), each row therefore has the same columns(attributes).
Tuple	It is a row in the relation.
Attribute	It is a column in the relation.
Degree of a relation	Number of columns in the relation
Cardinality of a relation	Number of rows in the relation
N-ary relation	Relation with degree N.
Domain	Set of allowed values for each attribute.

Example of a Relation

The instructor relation

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

attributes
(or columns)

tuples
(or rows)

Account

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

Basic Structure

Formally, given sets $D_1, D_2, \ldots. D_n$ 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 $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

Example: 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* x *customer_street* x *customer_city*

Attribute Types

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

Relation Schema

$A_1, A_2, \ldots, A_n$ are *attributes*

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

Example:

Customer_schema = (*customer_name, customer_street, customer_city*)

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

Example:

customer (Customer_schema)

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

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *account* relation with unordered tuples

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

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

  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.,if two customers own an account (What gets repeated?)
  - the need for null values
    - e.g., to represent a customer without an account

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

Keys

Let $K \subseteq R$, *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)* by "possible *r* " we mean a relation *r* that could exist in the enterprise we are modeling.

- Example: {*customer_name, customer_street*} and {*customer_name*} are both superkeys of *Customer*, if no two customers can possibly have the same name
  - In real life, an attribute such as *customer_id* would be used instead of *customer_name* to uniquely identify customers, but we omit it to keep our examples small, and instead assume customer names are unique.

Keys (Cont.)

- *K* is a **candidate key** if *K* is minimal

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

- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation
  - Should choose an attribute whose value never, or very rarely, changes.
  - E.g. Email address is unique, but may change

Keys…

Answer the following by understanding the requirements given below.

CUSTOMER(Custid, Name,Mid_Name,LastName, City, phone, email)

ACCOUNT(AccNo, CustId,Intr_CustId, AccType, Branch)

In Bank every customer will have Unique **CustomerID**. **Intr_CustId** is the Customer Id of customer who is introducing a new customer to the Bank.

A customer can have multiple accounts such as SB, Current, Loan etc. Every **Accno** is unique. **Name , Mid_name and Last_Name** information about a customer must be distinguishable from other customers. **Phone** - phone number of the customer. **Email-** Email Id of the customer. Every Customer has a unique phone number and email id.

Keys

- Is **(Phone, Email)** is a Super Key , if yes is it a minimal Super key ?
  - **K= (Phone, Email), Super key –YES**
  - **Is K minimal Super Key?**
    - **K - Phone ={Email} .** Another possibility is **K - Email ={ Phone}**
    - **Email or Phone alone can be used to identify every tuple uniquely, hence K is not minimal.**
  - **Is Email a minimal super key?**
    - **Email is minimal Super key & hence it is a Candidate key.**
  - Is Phone a minimal super key?
    - **Phone is minimal Super key & hence it is a Candidate key.**
  - **In this case Phone, Email & even CustId are candidate keys**
- **Is (Name,Mid_Name,LastName) a Super Key or a candidate key?**

Keys (Cont’d)

- Is **(AccNo, CustId)** a Super Key in ACCOUNT relation ? If yes ,is it a minimal Super key ?
- Yes, K= **(AccNo, CustId)** is a super key in ACCOUNT
- K – AccNo=Custid can take duplicate values because a customer can have multiple accounts so alone cannot be used to identify the tuples in Account relation
- K-Custid=Accno alone can be used to identify the tuples in Account relation
- Hence K is not a minimal super key.
- But Accno is a minimal super key or candidate key

Keys (Cont'd)

- Is (Custid, Name, Mid_Name,LastName) a Super Key in CUSTOMER Relation?
- If Yes, is it minimal Super key ?
- List possible minimal Super keys (Candidate Keys)

- K= (Custid, Name, Mid_Name, LastName) is a super key in CUSTOMER relation
- Among all the proper subsets (custid) takes unique values, therefore

(Custid, Name, Mid_Name,LastName) cannot be a minimal super key

Keys

A relation may have **multiple minimal Super Keys** (Candidate Key)

One of them may be considered as Primary Key

- Ex: **Cust_Id** in Customer may be Primary Key

Remaining all Candidate Keys are called as **Alternate Keys**

Foreign Keys

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

Account

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

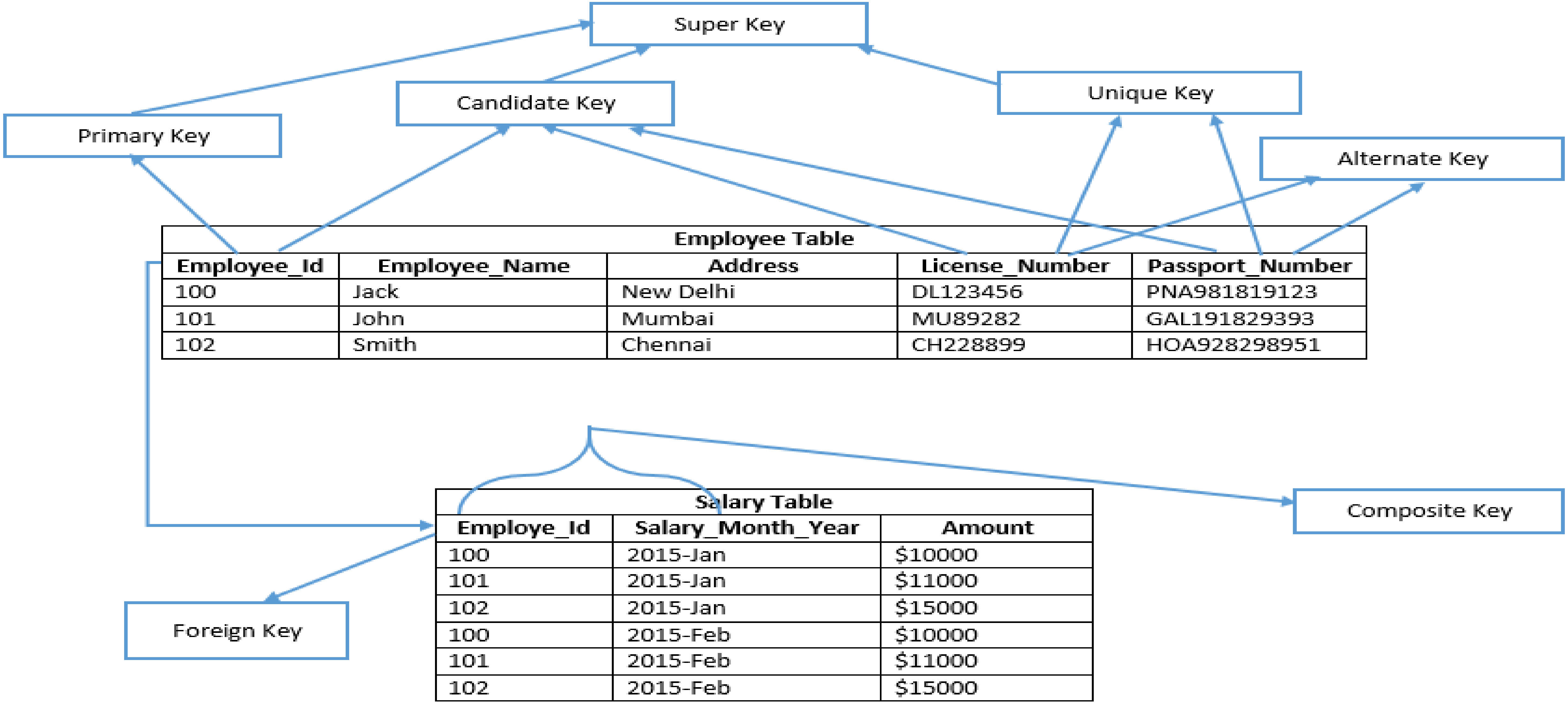
customer_name	account_number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Depositor

Foreign Keys

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

Types of Keys



Employee Table				
Employee_Id	**Employee_Name**	**Address**	**License_Number**	**Passport_Number**
100	Jack	New Delhi	DL123456	PNA981819123
101	John	Mumbai	MU89282	GAL191829393
102	Smith	Chennai	CH228899	HOA928298951

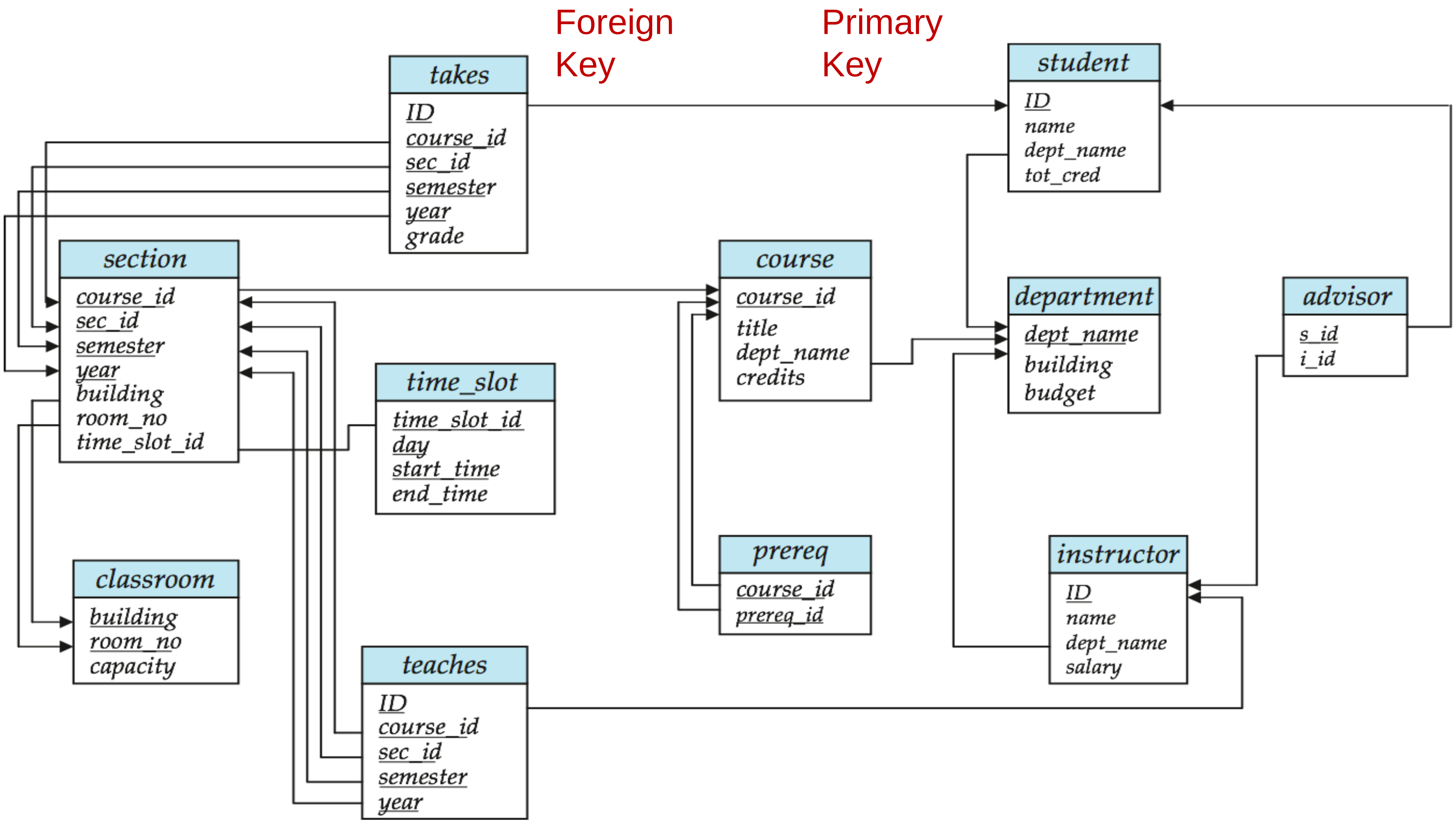
Salary Table		
Employe_Id	**Salary_Month_Year**	**Amount**
100	2015-Jan	$10000
101	2015-Jan	$11000
102	2015-Jan	$15000
100	2015-Feb	$10000
101	2015-Feb	$11000
102	2015-Feb	$15000

Schema Diagram

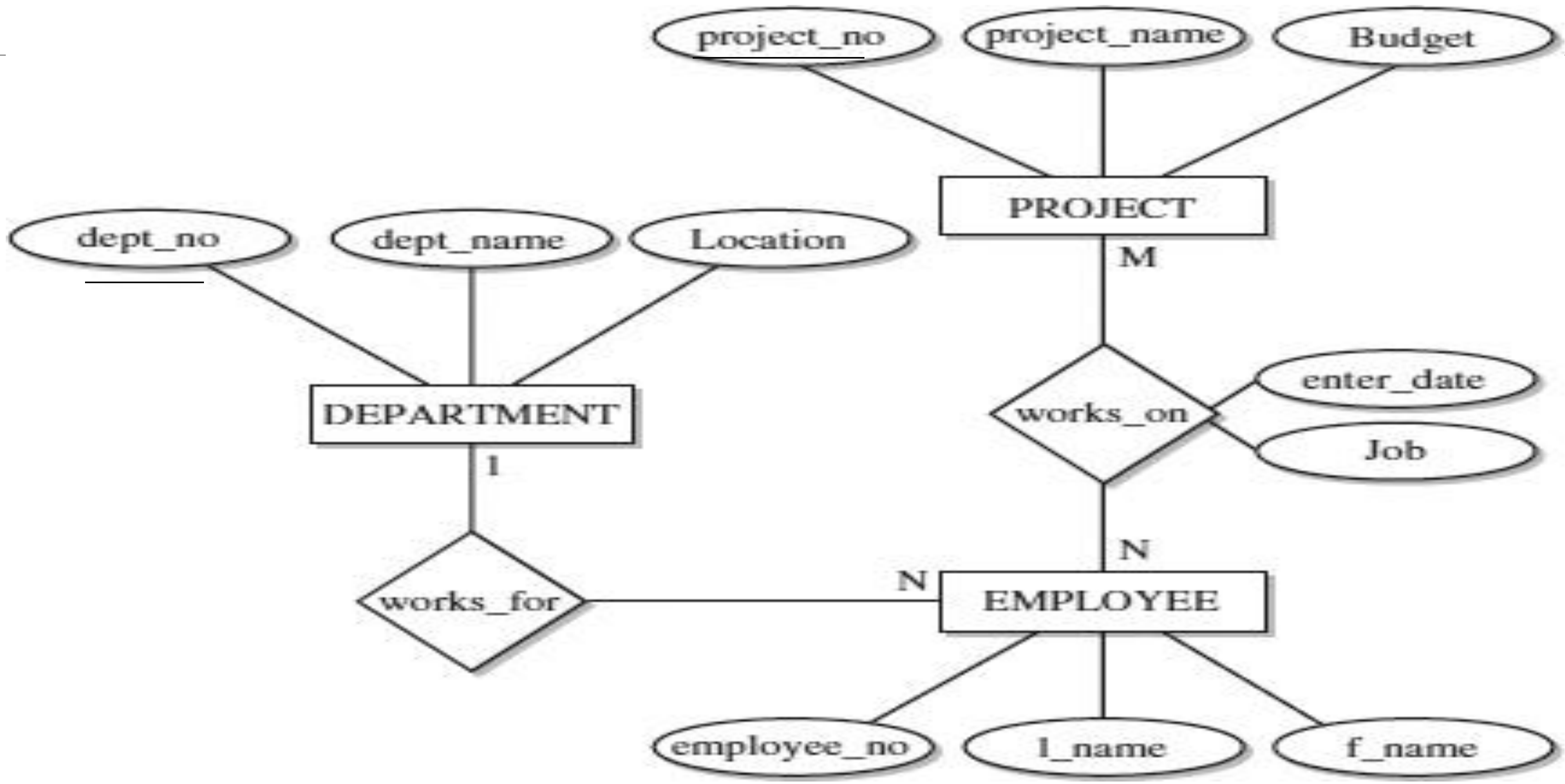
- A database schema along with primary key and foreign key dependencies can be represented by schema diagram.

  - Relation appears as box.
  - Relation name at the top
  - Attributes listed inside the box.
  - Primary key attributes are underlined.
  - Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

Schema Diagram for University Database



Convert the ER Diagram to Schema Diagram



Relational Algebra

- Procedural language
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

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

Select Operation

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

account

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

Defined as:

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

Where *p* is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

Each **term** is one of:

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

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

Example of selection:

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

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

Project Operation

Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

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

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

Example: To display the details of account number along with the balance amount

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

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

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

Union Operation

Notation: $r \cup s$

Defined as:

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

For $r \cup s$ to be valid.

1. *r, s* must have the *same* **arity** (same number of attributes)
2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

Example: to find all customers with either an account or a loan

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

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

Depositor

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

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

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

Example: to find all customers with who has an account but not loan

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

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

Cartesian-Product Operation

Notation $r \times s$

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, $R \cap S = \varnothing$).

If attributes of *r(R)* and *s(S*) are not disjoint, then renaming must be used.

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

Composition of Operations

Can build expressions using multiple operations

Example: $\sigma_{A=C}(r \; x \; s)$

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

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

A	B	C	D	E
α	1	α	10	a
β	2	β	10	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 $\rho_{x(A_1, A_2, \ldots, A_n)}(E)$

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

Union, Intersection & Set Difference

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

Two conditions

*r and s must be of the **same arity**

*$\mathbf{I^{th}}$ attribute in r and s must be from **same domain**

- Union:

r ∪ s

A	B
α	1
α	2
β	1
β	3

- Intersection

r ∩ s:

A	B
α	2

Is this true ?

$$r \cap s = r - (r - s)$$

- Set Difference

r - s:

A	B
α	1
β	1

Joining two relations – Natural Join

- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, the "natural join" of relations *R* and *S* is a relation on schema ***R* ∪ *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***

A	B	C	D
α	1	α	a
β	2	γ	a

r

B	D	E
1	a	α
3	a	β

s

A	B	C	D	E
α	1	α	a	α

Equating attributes of the same name, and Projecting out one copy of each pair of equated attributes

Natural Join 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	ε

s

Cartesian product followed by SELECT(σ) operation. Selection is based on equality on common Attributes in both relations. Finally removes duplicate attributes

- **Natural Join**
  - r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Consider two relations *r*(*R*) and *s*(*S*).
***R* ∩ *S* = {*A*1, *A*2, . . .,*An*}.**

$$r \bowtie s = \Pi_{R \cup S}\left(\sigma_{r.A_1 = s.A_1 \land r.A_2 = s.A_2 \land \ldots \land r.A_n = s.A_n}\ r \times s\right)$$

Quiz Q3: The natural join operation matches tuples (rows) whose values for common attributes are (1) not equal (2) equal (3) weird Greek letters (4) null

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

Instructor

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Department

Instructor $\bowtie$ Department

ID	*name*	*salary*	*dept_name*	*building*	*budget*
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
12121	Wu	90000	Finance	Painter	120000
15151	Mozart	40000	Music	Packard	80000
22222	Einstein	95000	Physics	Watson	70000
32343	El Said	60000	History	Painter	50000
33456	Gold	87000	Physics	Watson	70000
45565	Katz	75000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
76543	Singh	80000	Finance	Painter	120000
76766	Crick	72000	Biology	Watson	90000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000

Figure 2.12 Result of natural join of the *instructor* and *department* relations.

THETA JOIN

- The *theta join* operation is a variant of 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(condition) on attributes in the schema *R* ∪ *S*.
- The **theta join** operation ***r*** ***s*** is defined as follows:

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

It is equivalent to-

- Take the product **r X s**.
- Then apply $\boldsymbol{\sigma_\theta}$ to the result.

As for **σ, ϴ** can be any Boolean-valued condition. Historic versions of this operator allowed only A θ B, **where θ is =, <, etc**.; hence the name "theta-join."

Banking Example

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

Find all loans of over $1200

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

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

- Find the loan number for each loan of an amount greater than $1200 and less than $2000

$$\Pi_{loan\_number}\ (\sigma_{amount\ >\ 1200\ \wedge\ amount\ <2000}\ (loan))$$

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

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

Bank Example Queries

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

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

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

Bank Example Queries

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

$\Pi_{customer\_name}\ (\sigma_{branch\_name\ =\ \text{“Downtown”}}\ (\ (borrower \bowtie loan)))$

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

Find the names of all customers who have a balance amount >500 $

$$\Pi_{customer\_name}(\sigma_{balance > 500}(Account) \bowtie Depositor)$$

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

Account

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

Depositor

Example Queries

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.

$$\Pi_{customer\_name}(\sigma_{branch\_name=\text{"Perryridge"}}(borrower \bowtie loan)) - \Pi_{customer\_name}(depositor)$$

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

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

Depositor

Example Queries

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

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

Formal Definition

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

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 always be made to a temporary relation variable.

Assignment Operation

Example: $temp1 \leftarrow \prod_{R-S}(r)$

$temp2 \leftarrow \prod_{R-S}((temp1 \times s) - \prod_{R-S,S}(r))$

$result = temp1 - temp2$

Bank Example Queries

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

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

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

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

Depositor

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

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

E is any relational-algebra expression/ a relation

  - ***G_1, G_2 …, G_n*** is a list of attribute/s on which to group (can be empty)
  - Each ***F_i*** is an aggregate function
  - Each ***A_i*** is an attribute name on which aggregate function applied.

- Note: Some books/articles use γ (gamma) instead of $\mathcal{G}$ (Calligraphic G)

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

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

sum(*c*)
27

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

A	sum(*c*)
α	14
β	13

- What is the result for the following expression ?

$_{A,B}\mathcal{G}_{\text{sum(c)}}(r)$

Aggregate Operation – Example

- Find the average salary in each department

$$_{dept\_name}\mathcal{G}_{avg(salary)}(instructor)$$

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

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Outer Join

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

Relation *loan*

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

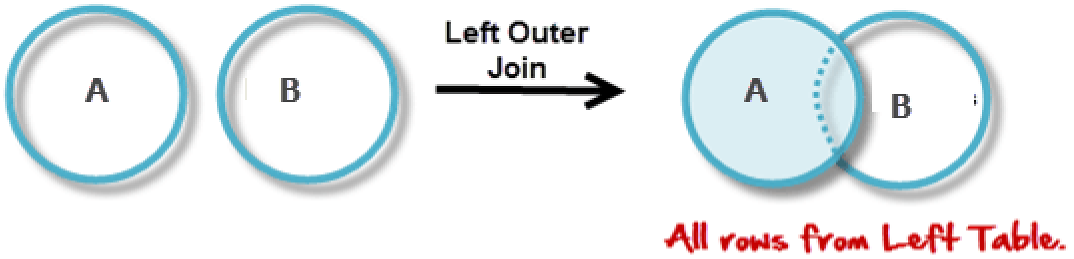
- Relation *borrower*

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

Outer Join – Example

Join

loan ⋈ borrower



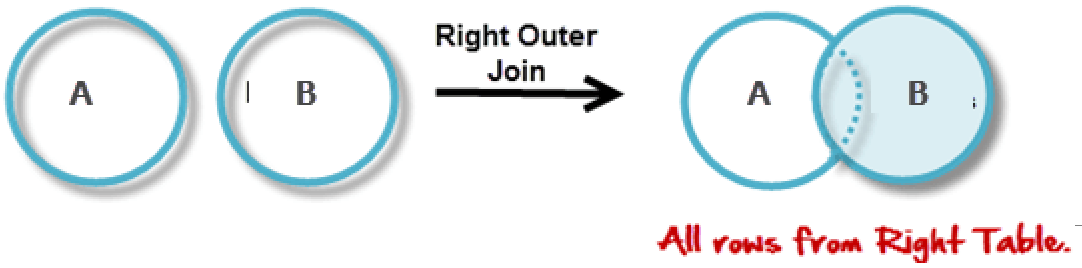
loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- **Left Outer Join**

loan ⟕ borrower

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

Outer Join – Example



- Right Outer Join

  loan ⟖ *borrower*

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

- **Full Outer Join**

  loan ⟗ ***borrower***

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

Composition of Relational Operations

“Find the names of all instructors in the Physics department.”

$$\Pi_{name}\,(\sigma_{dept\ name\ =\text{“Physics”}}\,(instructor))$$

Exercise

Consider the following Student database:

Student(Id,Stdname,City,tutor)

Enrolment(Id,Code,Marks)

Course(Code,title,Department)

Write the following queries in relational algebra ,assume that the attribute tutor is the Id of the tutor.

a)List all courses offered by Computer department

b)List the names of the students along with the names of the course opted.

c)List the names of students who have scored >80 marks.

d)List the details of students who have taken up the course offered by Physics department

e)Display the Student Id along with the total marks

a) $\pi_{code,\ title}(\sigma_{Department = \text{"computer"}}(course))$

b) $\pi_{stdname,\ title}((student \bowtie enrolment) \bowtie course)$

c) $\pi_{stdname}(\sigma_{marks > 80}(student \bowtie enrolment))$

d) $\pi_{ID,\ stdname}((\sigma_{Department = \text{"physics"}}(course) \bowtie enrolment) \bowtie student)$

e) ${}_{ID}\mathcal{G}_{sum(marks)}(enrolment)$

Exercise

Consider a relations **A(ID, Name, Age) , B(EID, Phone, City) ,**

C(ID, Salary, DeptName)

Note: EID and ID derived from Same domain

Write a relational Algebraic expression –

- To find ID, Name, Phone of Employees.
- To find Name of Employees having **Salary>90000**
- To find DeptName and total salary of each department.
- To find Name of Employees who from city - **Manipal**

Solution: A(ID, Name, Age) , B(EID, Phone, City) , C(ID, Salary, DeptName)

Note: EID and ID derived from Same domain

- To find ID, Name, Phone of Employees.
  - **PROJECT$_{\textbf{ID, Name, Phone}}$ (A Theat$_{\textbf{ID=EID}}$ B)**
- To find Name of Employees having **Salary>90000**
  - **PROJECT $_{\textbf{Name}}$ (SELECT $_{\textbf{Salary>90000}}$ (A N.JOIN C))**

    other way is
  - **PROJECT $_{\textbf{Name}}$ ((SELECT $_{\textbf{Salary>90000}}$ (C)) N.JOIN A)**
- To find DeptName and total salary of each department.
  - **$_{\textbf{DeptName}}$ G $_{\textbf{SUM(Salary)}}$ (C)**
- To find Name of Employees who from city - **Manipal**
  - **PROJECT** $_{\text{Name}}$ (A **Theta** $_{\textbf{(ID=EID AND City='Manipal')}}$ B)

END