SNS COLLEGE OF TECHNOLOGY

COIMBATORE

AN AUTONOMOUS INSTITUTION

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE New Delhi & affiliated to the Anna University, Chennai

DEPARTMENT OF MCA

Course Name : 19CAT609 - DATA BASE MANAGEMENT SYSTEM

Class : I Year / I Semester

Unit II - Introduction

Topic I – Relational Model

Relational Model

- Data Representation
- The way integrity constraints expressed?
- Data creation, management and manipulation
- Extended Relational-Algebra-Operations
- Database design
- Data independence

Example of a Relation

- Main construct for representing data in the relational model is a relation.
- A relation consists of a relation schema and a relation instance
- A relation schema describes the column heads for the table
- Students(sid: string, name: string, login: string, age: integer, gpa: real)
- An instance of a relation is a set of tuples, also called records
- A relation instance can be thought of as a table in which each tuple is a row, and all rows have the same number of fields
- Degree, also called arity, of a relation is the number of fields. The cardinality of a relation instance is the number of tuples in it

Example of a Relation

FIELDS (ATTRIBUTES, COLUMNS)

Field names

TUPLES

(RECORDS, ROWS)

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Relation schema

- It specifies the domain of each field or column in the relation instance
- domain constraints in the schema specify an important condition that we want each instance of the relation to satisfy
- Domain of a field is essentially the type of that field

Basic Structure

- let R(f1:D1, ..., fn:Dn) be a relation schema,
- For each fi, 1 ≤ i ≤ n, let Domi be the set of values associated with the domain named Di
- An instance of R that satisfies the domain constraints in the schema is a set of tuples with n fields

$$\{ \langle f_1 : d_1, \ \ldots \ , f_n : d_n \rangle \ | \ d_1 \in Dom_1, \ \ldots \ , d_n \in Dom_n \}$$

- angular brackets h...i identify the fields of a tuple
- {...} denote a set (of tuples)
- vertical bar | should be read ‘such that,’ the symbol ∈ should be read ‘in,

Relation schema

A relational database is a collection of relations with distinct relation names

Relational database schema is the collection of schemas for the relations in the database

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)

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. For instance

  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
- Normalization theory deals with how to design relational schemas

The customer Relation

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

The depositor Relation

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

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}

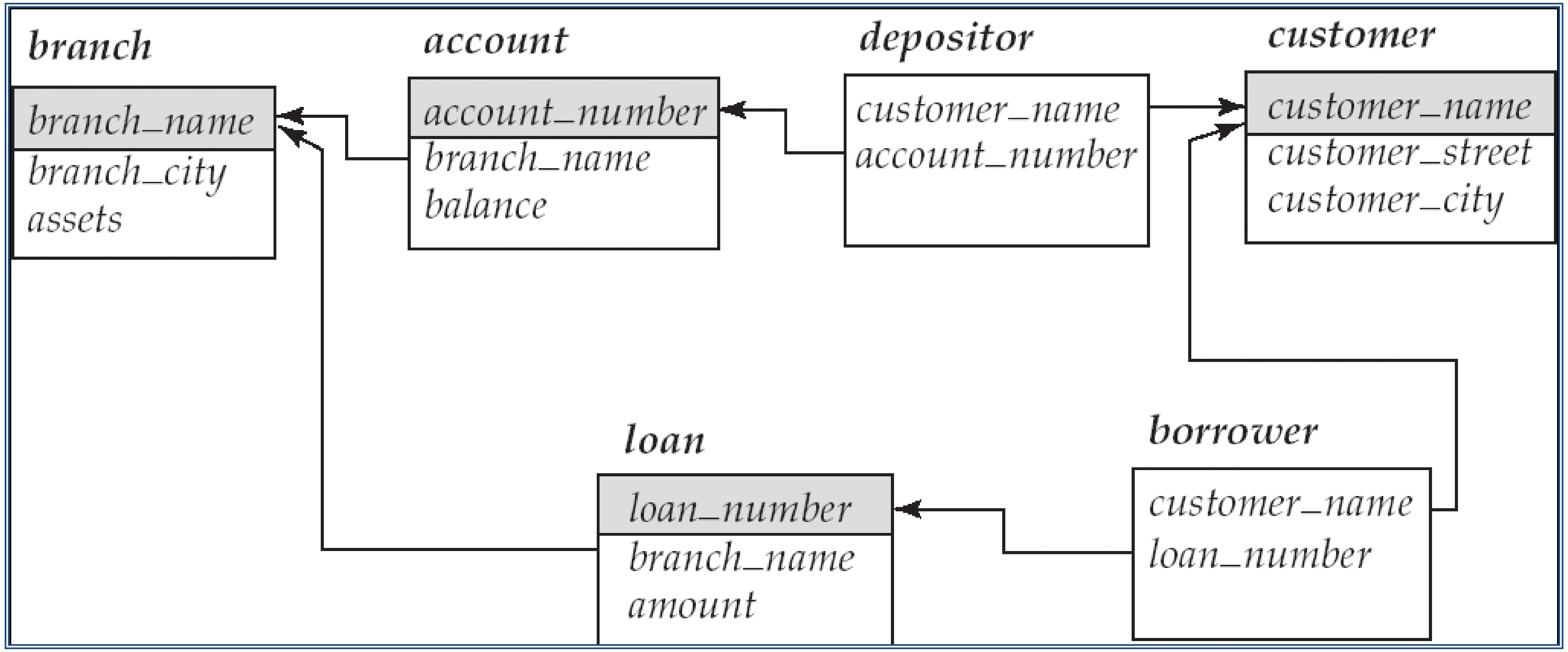
  are both superkeys of Customer, if no two customers can possibly have the same name
- In real life, an attribute such as customer_id would be used instead of customer_name to uniquely identify customers

Keys

- K is a **candidate key** if K is minimal
  Example: {customer_name} is a candidate key for Customer, since it is a superkey and no subset of it is a superkey.
- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation
- Should choose an attribute whose value never, or very rarely, changes.
- E.g. email address is unique, but may change

Foreign 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.



Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
  - “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
  - Pure languages form underlying basis of query languages that people use.

Relational Algebra

Procedural language

Six basic operators

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

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

Select Operation

- Relation r

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

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

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

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

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

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

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

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

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

Project Operation – 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
α
β
β

=

C
1
1
2

Project Operation

Notation:

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

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

Duplicate rows removed from result, since relations are sets

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

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

Union Operation – Example

- 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

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

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

Union Operation – Example

- Relations r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r – s:

A	B
α	1
β	1

Set Difference Operation

Notation $r - s$

Defined as:

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

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

- *r* and *s* must have the same arity
- attribute domains of *r* and *s* must be compatible

Cartesian-Product Operation – Example

Relations r, s:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

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

Cartesian-Product Operation

Notation $r \times s$

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

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

Composition of Operations

Can build expressions using multiple operations

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

r x s

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

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

A	B	C	D	E
α	1	α	10	a
β	2	β	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 returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots., A_n$.

Banking Example

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)

Example Queries

- Find all loans of over $1200

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

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

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

- 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

- 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

- 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.
  - –The query is:

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

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:

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_p (E_1)$, 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

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:

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_p (E_1)$, 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

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- ✓ Set intersection
- ✓ Natural join
- ✓ Division
- ✓ Assignment

Set-Intersection Operation

Notation: $r \cap s$

Defined as:

$r \cap s = \{ t \mid t \in r$ **and** $t \in s \}$

Assume:

- r, s have the *same arity*
- attributes of r and s are compatible

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

Set-Intersection Operation – Example

Relation *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

Natural-Join Operation

Notation: r ⋈ s

Let *r* and *s* be relations on schemas *R* and *S* respectively.

Then, r ⋈ s is a relation on schema $R \cup S$ obtained as follows:

- Consider each pair of tuples t_r from *r* and t_s from *s*.
- If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
  - *t* has the same value as t_r on *r*
  - *t* has the same value as t_s on *s*

Example:

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

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

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	$\in$

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

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example: EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

Join Operations

Example: SALARY

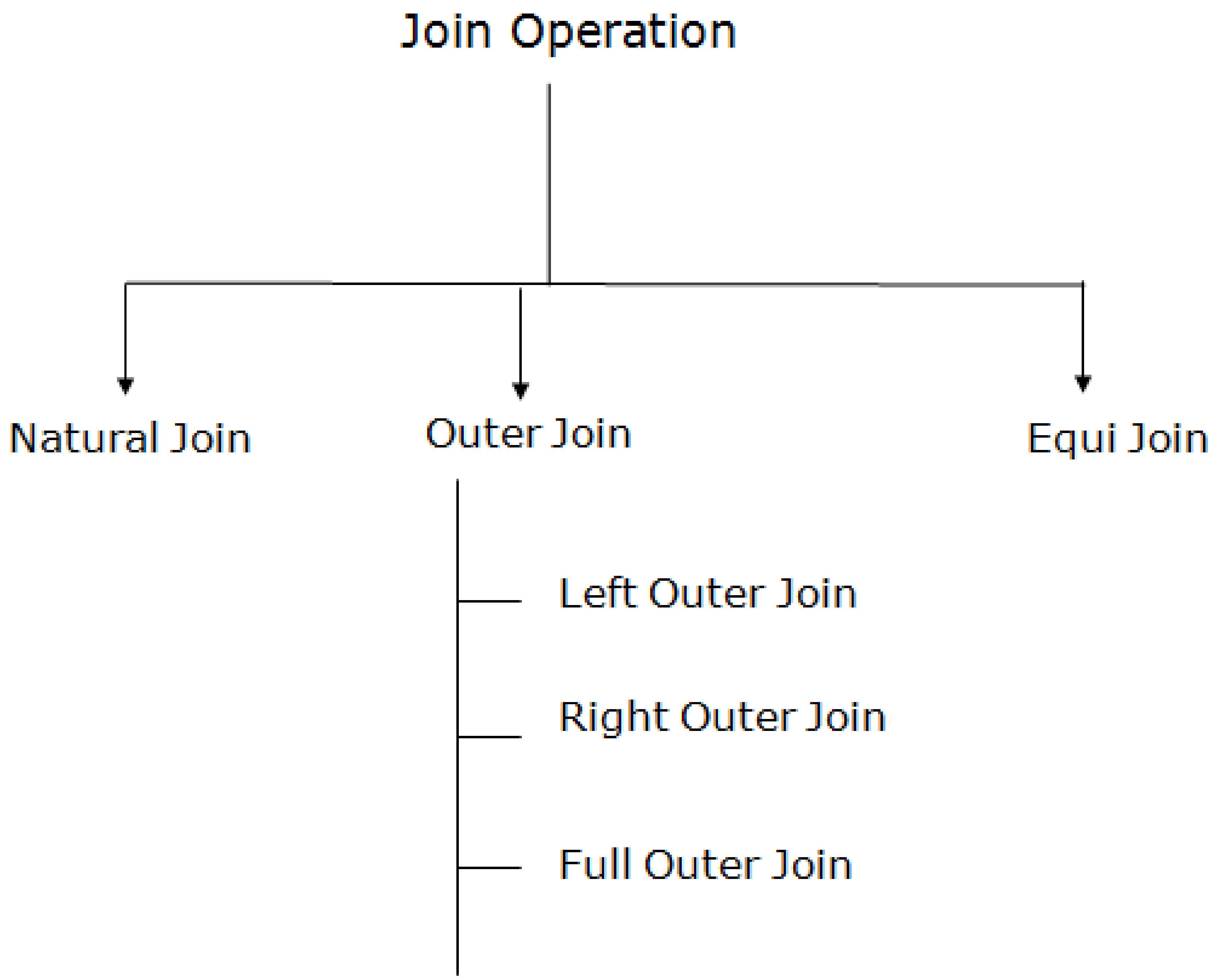
EMP_CODE	SALARY
101	50000
102	30000
103	25000

Example: SALARY

Operation: (EMPLOYEE ⋈ SALARY)

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations



1. Natural Join

A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.

It is denoted by ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example: EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

2. Outer Join

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input:

(EMPLOYEE ⋈ FACT_WORKERS)

2. Outer Join

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

Outer Join

An outer join is basically of three types:

- Left outer join
- Right outer join
- Full outer join

Outer Join

a. Left outer join:

Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.

In the left outer join, tuples in R have no matching tuples in S.

It is denoted by ⟕.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input: EMPLOYEE ⟕ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

Outer Join

b. Right outer join:

Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.

In right outer join, tuples in S have no matching tuples in R.

It is denoted by ⟖.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input: EMPLOYEE ⟖ FACT_WORKERS

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

Outer Join

c. Full outer join:

Full outer join is like a left or right join except that it contains all rows from both tables.

In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name. It is denoted by ⟗.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input: EMPLOYEE ⟗ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example: CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

3. Equi join

Input: CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Division Operation

Notation:

Suited to queries that include the phrase "for all".

Let *r* and *s* be relations on schemas *R* and *S* respectively where

$R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$

$S = (B_1, \ldots, B_n)$

The result of r ÷ s is a relation on schema

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

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

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

Division Operation – 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

Another Division Example

- Relations r, s:

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

r

D	E
a	1
b	1

s

- $r \div s$:

A	B	C
α	a	γ
γ	a	γ

Division Operation (Cont.)

Property

Let $q = r \div s$

Then q is the largest relation satisfying $q \times s \subseteq r$

Definition in terms of the basic algebra operation

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

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

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 ($\leftarrow$) 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.

Example: Write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}(r)$$

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

$$result = temp1 - temp2$$

The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.

May use variable in subsequent expressions.

Bank Example Queries

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

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

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

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

Bank Example Queries

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

- Query 1

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

- Query 2

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

Note that Query 2 uses a constant relation

Bank Example Queries

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

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

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join

Generalized Projection

Extends the projection operation by allowing arithmetic functions to be used in the projection list

E is any relational-algebra expression

Each of $F_1, F_2, \ldots, F_n$ are are arithmetic expressions 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

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

E is any relational-algebra expression

- $G_1, G_2, \ldots, G_n$ is a list of attributes on which to group (can be empty)
- Each F_i is an aggregate function
- 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

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

sum(*c*)
27

Aggregate Operation – Example

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

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

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

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

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

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

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.
  - We shall study precise meaning of comparisons with nulls later

Outer Join

- 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

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Null Values

Comparisons with null values return the special truth value: *unknown*

If *false* was used instead of *unknown*, then *not (A < 5)*

would not be equivalent to *A >= 5*

Three-valued logic using the truth value *unknown*:

OR: (*unknown* **or** *true*) = *true*,

(*unknown* **or** *false*) = *unknown*

(*unknown* **or** *unknown*) = *unknown*

AND: (*true* **and** *unknown*) = *unknown*,

(*false* **and** *unknown*) = *false*,

(*unknown* **and** *unknown*) = *unknown*

NOT: (**not** *unknown*) = *unknown*

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 operator.

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 delete only whole tuples; cannot delete values on only 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

- 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

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

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{\ branch\_city\ =\ \text{“Needham”}}\ (account \bowtie branch\ )$$

$$\text{r}_2 \leftarrow \prod_{\ account\_number,\ branch\_name,\ balance}\ (r_1)$$

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

$$account \leftarrow \text{account} - r_2$$

$$depositor \leftarrow \text{depositor} - r_3$$

Insertion

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

$$account \leftarrow account \cup \{(\text{“A-973”}, \text{“Perryridge”}, 1200)\}$$

$$depositor \leftarrow depositor \cup \{(\text{“Smith”}, \text{“A-973”})\}$$

- 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 charging *all* values in the tuple

Use the generalized projection operator to do this task

Each F_i is either

- the I^{th} attribute of *r*, if the I^{th} attribute is not updated, or,
- if the attribute is 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.

$$account \leftarrow \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (account)$$

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

$$\begin{aligned} account \leftarrow\ & \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.06}\ (\sigma_{\,BAL\ >\ 10000}\ (account\ )) \\ & \cup\ \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (\sigma_{BAL\ \leq\ 10000}\ (account)) \end{aligned}$$

Figure 2.3. The branch relation

branch_name	*branch_city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

Figure 2.6: The *loan* relation

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

Figure 2.7: The *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

Figure 2.9 Result of $\sigma_{\text{branch\_name = “Perryridge”}}$ (*loan*)

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

Figure 2.10: Loan number and the amount of the loan

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

Figure 2.11: Names of all customers who have either an account or an loan

customer_name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Figure 2.12: Customers with an account but no loan

customer_name
Johnson
Lindsay
Turner

Figure 2.13: Result of *borrower* |X| *loan*

customer_name	*borrower. loan_number*	*loan. loan_number*	*branch_name*	*amount*
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-23	Redwood	2000
Adams	L-16	L-93	Mianus	500
Curry	L-93	L-11	Round Hill	900
Curry	L-93	L-14	Downtown	1500
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Curry	L-93	L-17	Downtown	1000
Curry	L-93	L-23	Redwood	2000
Curry	L-93	L-93	Mianus	500
Hayes	L-15	L-11		900
Hayes	L-15	L-14		1500
Hayes	L-15	L-15		1500
Hayes	L-15	L-16		1300
Hayes	L-15	L-17		1000
Hayes	L-15	L-23		2000
Hayes	L-15	L-93		500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Smith	L-23	L-11	Round Hill	900
Smith	L-23	L-14	Downtown	1500
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-23	Redwood	2000
Smith	L-23	L-93	Mianus	500
Williams	L-17	L-11	Round Hill	900
Williams	L-17	L-14	Downtown	1500
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300
Williams	L-17	L-17	Downtown	1000
Williams	L-17	L-23	Redwood	2000
Williams	L-17	L-93	Mianus	500

Figure 2.14

customer_name	borrower. loan_number	loan. loan_number	branch_name	amount
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Hayes	L-15	L-15	Perryridge	1500
Hayes	L-15	L-16	Perryridge	1300
Jackson	L-14	L-15	Perryridge	1500
Jackson	L-14	L-16	Perryridge	1300
Jones	L-17	L-15	Perryridge	1500
Jones	L-17	L-16	Perryridge	1300
Smith	L-11	L-15	Perryridge	1500
Smith	L-11	L-16	Perryridge	1300
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300

Figure 2.15

customer_name
Adams
Hayes

Figure 2.16

balance
500
400
700
750
350

Figure 2.17 Largest account balance in the bank

balance
900

Figure 2.18: Customers who live on the same street and in the same city as Smith

customer_name
Curry
Smith

Figure 2.19: Customers with both an account and a loan at the bank

customer_name
Hayes
Jones
Smith

Figure 2.20

customer_name	*loan_number*	*amount*
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-23	2000
Smith	L-11	900
Williams	L-17	1000

Figure 2.21

branch_name
Brighton
Perryridge

Figure 2.22

branch_name
Brighton
Downtown

Figure 2.23

customer_name	*branch_name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

Figure 2.24: The *credit_info* relation

customer_name	*limit*	*credit_balance*
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

Figure 2.25

customer_name	*credit_available*
Curry	250
Jones	5300
Smith	1600
Hayes	0

Figure 2.26: The *pt_works* 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

Figure 2.27 The *pt_works* relation after regrouping

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

Figure 2.28

branch_name	*sum of salary*
Austin	3100
Downtown	5300
Perryridge	8100

Figure 2.29

branch_name	*sum_salary*	*max_salary*
Austin	3100	1600
Downtown	5300	2500
Perryridge	8100	5300

Figure 2.30 The *employee* and *ft_works relations*

employee_name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

employee_name	*branch_name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

Figure 2.31

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

Figure 2.32

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*

Figure 2.33

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gates	*null*	*null*	Redmond	5300

Figure 2.34

employee_name	*street*	*city*	*branch_name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*
Gates	*null*	*null*	Redmond	5300

Reference

1. https://www.javatpoint.com/dbms-data-model-schema-and-instance
2. https://hirinfotech.com/structured-vs-unstructured-data/

THANK YOU