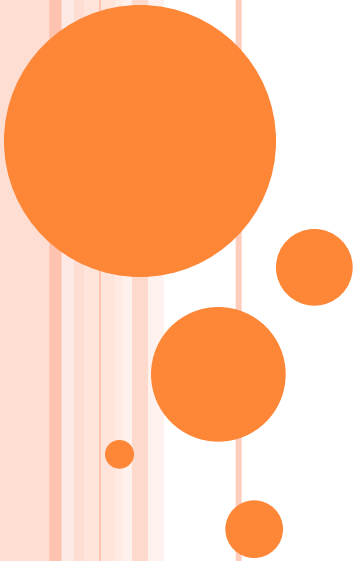
UNIT – 2

RELATIONAL MODEL



Relational Model Concepts

- The relational Model of Data is based on the concept of a Relation.
- A Relation is a mathematical concept based on the ideas of sets.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

INFORMAL DEFINITIONS

- RELATION: A table of values
  - A relation may be thought of as a **set of rows**.
  - A relation may alternately be though of as a **set of columns**.
  - Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
  - Each row has a value of an item or set of items that uniquely identifies that row in the table.
  - Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
  - Each column typically is called by its column name or column header or attribute name.

FORMAL DEFINITIONS

- A **Relation** may be defined in multiple ways.
- The **Schema** of a Relation: *R* (A1, A2, ...An)

  Relation schema *R* is defined over **attributes** A1, A2, ...An

  For Example -

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)

  Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

FORMAL DEFINITIONS

- A **tuple** is an ordered set of values
- Each value is derived from an appropriate domain.
- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.
- **<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">** is a tuple belonging to the CUSTOMER relation.
- A relation may be regarded as a ***set of tuples*** (rows).
- Columns in a table are also called attributes of the relation.

FORMAL DEFINITIONS

- A **domain** has a logical definition: e.g., "USA_phone_numbers" are the set of 10 digit phone numbers valid in the U.S.
- A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.
- An attribute designates the **role** played by the domain. E.g., the domain Date may be used to define attributes "Invoice-date" and "Payment-date".

FORMAL DEFINITIONS

- The relation is formed over the cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.
- For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.
- Formally,

  Given $R(A_1, A_2, ..., A_n)$

  $r(R) \subset dom(A_1) \times dom(A_2) \times ....\times dom(A_n)$
- R: schema of the relation
- r of R: a specific "value" or population of R.
- R is also called the **intension** of a relation
- r is also called the **extension** of a relation

FORMAL DEFINITIONS

- Let S1 = {0,1}
- Let S2 = {a,b,c}

- Let R $\subset$ S1 X S2

- Then for example: r(R) = {<0,a> , <0,b> , <1,c> }

  is one possible "state" or "population" or "extension" r of the relation R, defined over domains S1 and S2. It has three tuples.

DEFINITION SUMMARY

Informal Terms		Formal Terms
Table		Relation
Column		Attribute/Domain
Row		Tuple
Values in a column		Domain
Table Definition		Schema of a Relation
Populated Table		Extension

Relational Integrity Constraints

- Constraints are *conditions* that must hold on *all* valid relation instances. There are three main types of constraints:
  1. **Key** constraints
  2. **Entity integrity** constraints
  3. **Referential integrity** constraints

KEY CONSTRAINTS

- **Superkey** of R: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK. That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK].
- **Key** of R: A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.

Example: The CAR relation schema:

CAR(State, Reg#, SerialNo, Make, Model, Year)

has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but *not* a key.

- If a relation has *several* **candidate keys**, one is chosen arbitrarily to be the **primary key**. The primary key attributes are *underlined*.

KEY CONSTRAINTS

The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

Entity Integrity

- **Relational Database Schema**: A set S of relation schemas that belong to the same database. S is the *name* of the **database**.

$$S = \{R_1, R_2, ..., R_n\}$$

- **Entity Integrity**: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to *identify* the individual tuples.

$$t[PK] \neq \text{null for any tuple t in r(R)}$$

- Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- A constraint involving *two* relations (the previous constraints involve a *single* relation).
- Used to specify a *relationship* among tuples in two relations: the **referencing relation** and the **referenced relation**.
- Tuples in the *referencing relation* R_1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R_2. A tuple t_1 in R_1 is said to **reference** a tuple t_2 in R_2 if $t_1[FK] = t_2[PK]$.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from $R_1.FK$ to R_2.

Referential Integrity Constraint

Statement of the constraint

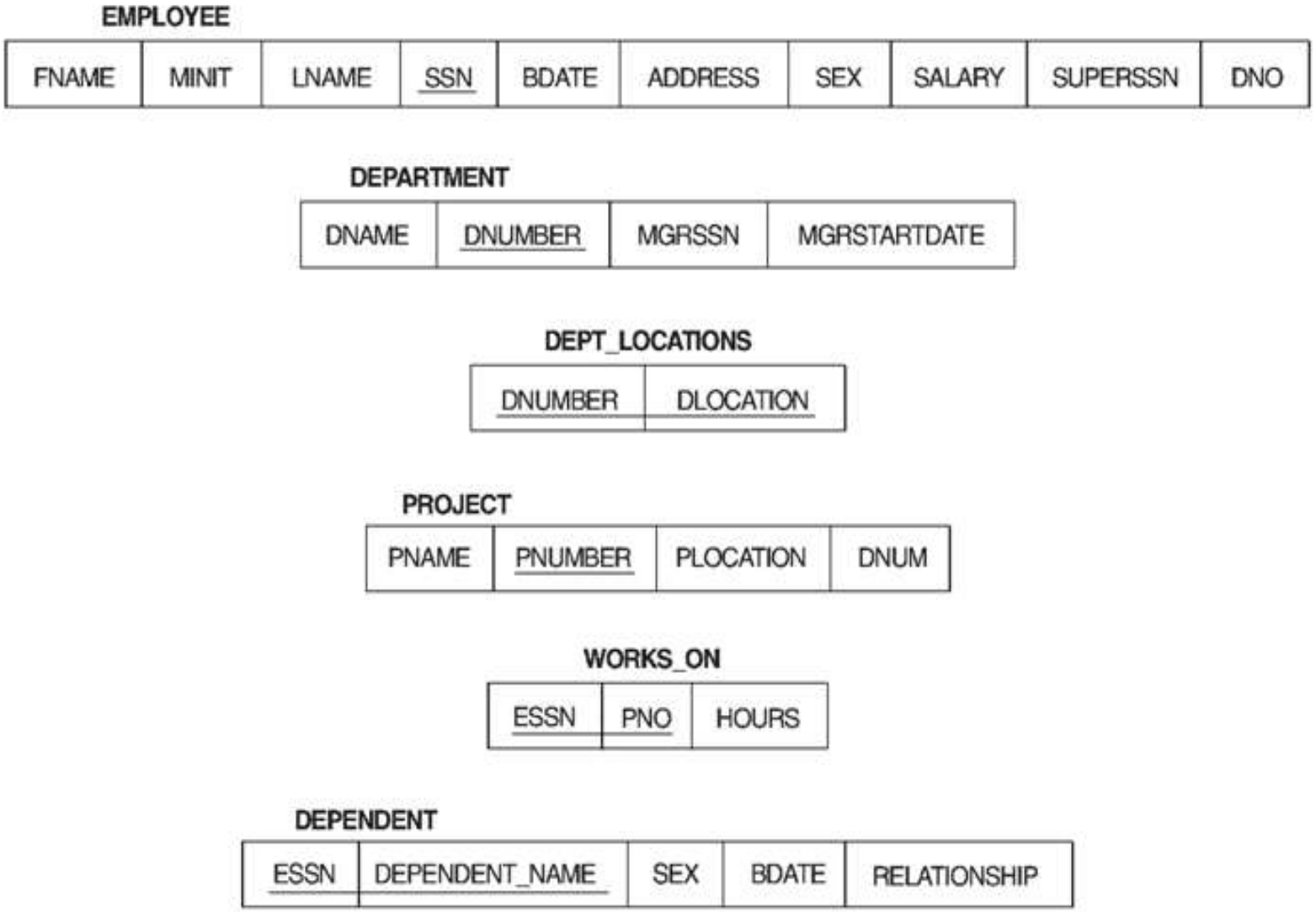
The value in the foreign key column (or columns) FK of the the **referencing relation** R_1 can be either:

(1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R_2, or..

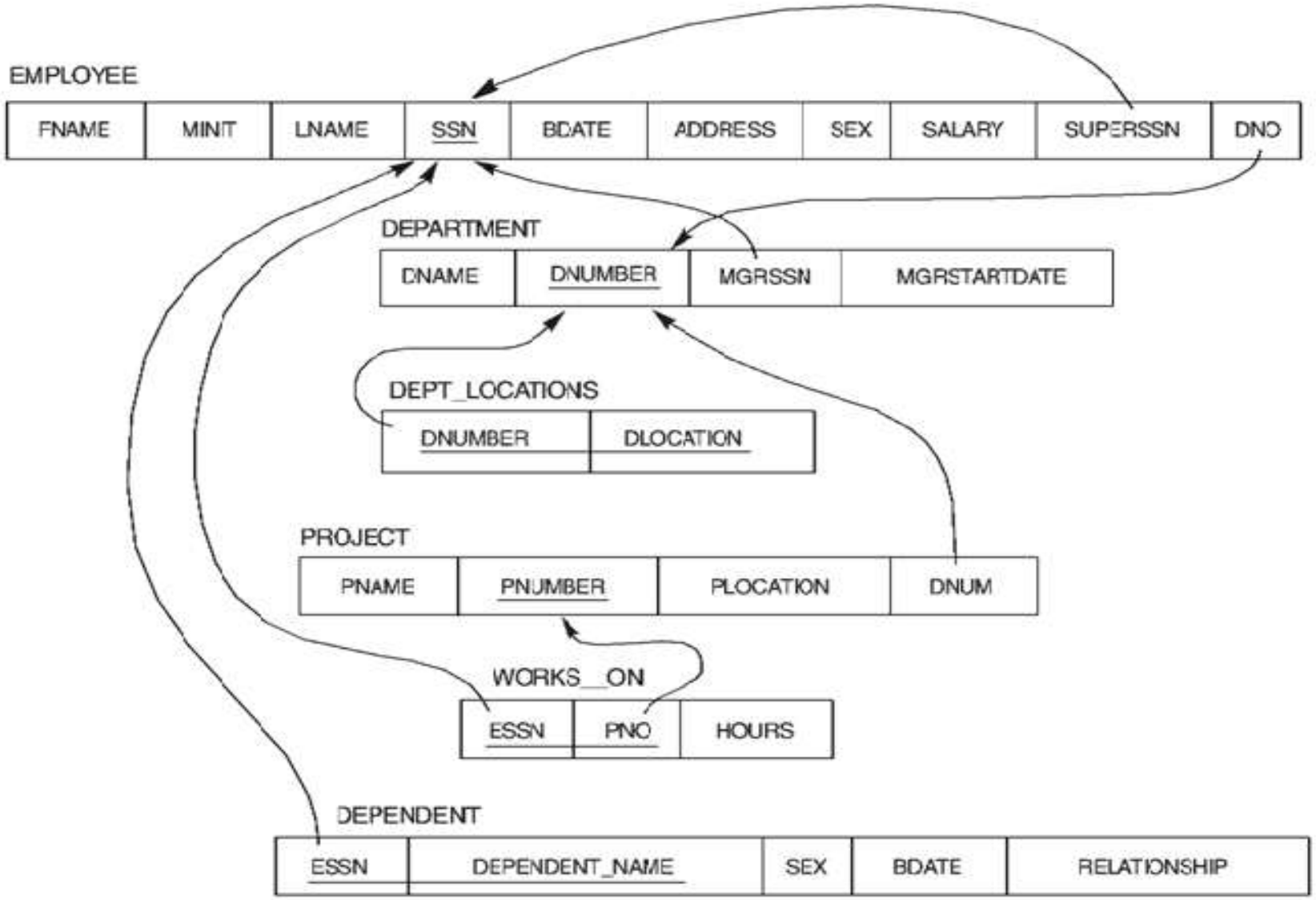
(2) a null.

In case (2), the FK in R_1 should not be a part of its own primary key.

Schema diagram for the COMPANY relational database schema; the primary keys are underlined.



Referential integrity constraints displayed on the COMPANY relational database schema diagram.



Relational Algebra

- The basic set of operations for the relational model is known as the relational algebra. These operations enable a user to specify basic retrieval requests.

- The result of a retrieval is a new relation, which may have been formed from one or more relations. The **algebra operations** thus produce new relations, which can be further manipulated using operations of the same algebra.

- A sequence of relational algebra operations forms a **relational algebra expression**, whose result will also be a relation that represents the result of a database query (or retrieval request).

UNARY RELATIONAL OPERATIONS

- **SELECT Operation**

SELECT operation is used to select a *subset* of the tuples from a relation that satisfy a **selection condition**. It is a filter that keeps only those tuples that satisfy a qualifying condition – those satisfying the condition are selected while others are discarded.

Example: To select the EMPLOYEE tuples whose department number is four or those whose salary is greater than Rs 30,000 the following notation is used:

σDNO = 4 (EMPLOYEE)

σSALARY > 30,000 (EMPLOYEE)

In general, the select operation is denoted by

σ <selection condition>(R)

where the symbol **σ** (sigma) is used to denote the select operator, and the selection condition is a Boolean expression specified on the attributes of relation R

UNARY RELATIONAL OPERATIONS

SELECT Operation Properties

- The SELECT operation $\sigma_{<\text{selection condition}>}(R)$ produces a relation S that has the same schema as R
- The SELECT operation σ is **commutative;** i.e.,
  $\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(R)) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition1}>}(R))$
- A cascaded SELECT operation **may be applied in any order;** i.e.,
  $\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(\sigma_{<\text{condition3}>}(R))$
  $= \sigma_{<\text{condition2}>}(\sigma_{<\text{condition3}>}(\sigma_{<\text{condition1}>}(R)))$
- A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions**;** i.e.,
  $\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(\sigma_{<\text{condition3}>}(R))$
  $= \sigma_{<\text{condition1}> \text{ AND } <\text{condition2}> \text{ AND } <\text{condition3}>}(R)))$

Unary Relational Operations (cont.)

- **PROJECT Operation**

This operation selects certain *columns* from the table and discards the other columns. The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.

Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

The general form of the project operation is

π<attribute list>(R)

where π (pi) is the symbol used to represent the project operation and <attribute list> is the desired list of attributes from the attributes of relation R.

The project operation *removes any duplicate tuples*, so the result of the project operation is a set of tuples and hence a valid relation.

UNARY RELATIONAL OPERATIONS (CONT.)

PROJECT Operation Properties

- The number of tuples in the result of projection $\pi_{\text{<list>}}(R)$ is always less or equal to the number of tuples in R.

- If the list of attributes includes a key of R, then the number of tuples is equal to the number of tuples in R.

- $\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$ as long as <list2> contains the attributes in <list1> ; otherwise, the left hand side is an incorrect expression.

Unary Relational Operations (cont.)

- **Rename Operation**

We may want to apply several relational algebra operations one after the other. Either we can write the operations as a single **relational algebra expression** by nesting the operations, or we can apply one operation at a time and create **intermediate result relations**. In the latter case, we must give names to the relations that hold the intermediate results.

Example: To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation. We can write a single relational algebra expression as follows:

$$\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$$

OR We can explicitly show the sequence of operations, giving a name to each intermediate relation:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$$

Unary Relational Operations (cont.)

- **Rename Operation (cont.)**

  The rename operator is ρ

  The general Rename operation can be expressed by any of the following forms:

- $\rho_{S(B_1, B_2, \ldots, B_n)}(R)$ is a renamed relation S based on R with column names $B_1, B_2, \ldots, B_n$.
- $\rho_S(R)$ is a renamed relation S based on R (which does not specify column names).
- $\rho_{(B_1, B_2, \ldots, B_n)}(R)$ is a renamed relation with column names $B_1, B_2, \ldots, B_n$ which does not specify a new relation name.

Relational Algebra Operations From Set Theory

- **UNION Operation**

  The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

R

A	B	C
1	5	X
5	10	AA
7	12	B

S

D	E	F
10	5	A
6	4	B
2	3	C
1	5	X

A	B	C
1	5	X
10	5	A
5	10	AA
6	4	B
7	12	B
2	3	C

Relational Algebra Operations From Set Theory (cont.)

- **INTERSECTION OPERATION**

The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S. The two operands must be "type compatible"

A	B	C
1	5	X

Relational Algebra Operations From Set Theory (cont.)

- **Set Difference (or MINUS) Operation**

The result of this operation, denoted by R - S, is a relation that includes all tuples that are in R but not in S. The two operands must be "type compatible".

Example: The figure shows the names of students who are not instructors, and the names of instructors who are not students.

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

STUDENT-INSTRUCTOR

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

INSTRUCTOR-STUDENT

Illustrating the set operations union, intersection, and difference. (a) Two union compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∪ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

CROSS PRODUCT

R * S

A	B	C	D	E	F
1	5	X	10	5	A
1	5	X	6	4	B
1	5	X	2	3	C
1	5	X	1	5	X
5	10	AA	10	5	A
5	10	AA	6	4	B
5	10	AA	2	3	C
5	10	AA	1	5	X
7	12	B	10	5	A
7	12	B	6	4	B
7	12	B	2	3	C
7	12	B	1	5	X

Binary Relational Operations

- **JOIN Operation**
  - The sequence of cartesian product followed by select is used quite commonly to identify and select related tuples from two relations, a special operation, called **JOIN**. It is denoted by a
  - This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.
  - The general form of a join operation on two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is:

$\bowtie$

$$R \bowtie_{\text{<join condition>}} S$$

where R and S can be any relations that result from general *relational algebra expressions.*

Binary Relational Operations (cont.)

Example: Suppose that we want to retrieve the name of the manager of each department. To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple. We do this by using the join operation.

DEPT_MGR ← DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

DEPT_MGR	DNAME	DNUMBER	MGRSSN	. . .	FNAME	MINIT	LNAME	SSN	. . .
	Research	5	333445555	. . .	Franklin	T	Wong	333445555	. . .
	Administration	4	987654321	. . .	Jennifer	S	Wallace	987654321	. . .
	Headquarters	1	888665555	. . .	James	E	Borg	888665555	. . .

FIGURE 6.6 Result of the JOIN operation DEPT_MGR ← DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE.

Binary Relational Operations (cont.)

- **EQUIJOIN Operation**

  The most common use of join involves join conditions with equality comparisons only. Such a join, where the only comparison operator used is =, is called an EQUIJOIN. In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have *identical values* in every tuple.

  The JOIN seen in the previous example was EQUIJOIN.

- **NATURAL JOIN Operation**

  Because one of each pair of attributes with identical values is superfluous, a new operation called natural join—denoted by *—was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.

  ⋈ The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the **same name** in both relations. If this is not the case, a renaming operation is applied first.

Binary Relational Operations (cont.)

(a)

PROJ_DEPT	PNAME	PNUMBER	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	ProductX	1	Bellaire	5	Research	333445555	1988-05-22
	ProductY	2	Sugarland	5	Research	333445555	1988-05-22
	ProductZ	3	Houston	5	Research	333445555	1988-05-22
	Computerization	10	Stafford	4	Administration	987654321	1995-01-01
	Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
	Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	LOCATION
	Headquarters	1	888665555	1981-06-19	Houston
	Administration	4	987654321	1995-01-01	Stafford
	Research	5	333445555	1988-05-22	Bellaire
	Research	5	333445555	1988-05-22	Sugarland
	Research	5	333445555	1988-05-22	Houston

FIGURE 6.7 Results of two NATURAL JOIN operations. (a) PROJ_DEPT ← PROJECT * DEPT. (b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

Complete Set of Relational Operations

- The set of operations including **select σ, project π , union ∪, set difference - , and cartesian product X** is called a complete set because any other relational algebra expression can be expressed by a combination of these five operations.

- For example:

  $\mathbf{R \cap S = (R \cup S) – ((R – S) \cup (S – R))}$

  $R \bowtie_{\text{<join condition>}} S = \sigma_{\text{<join condition>}} (R \times S)$

Binary Relational Operations (cont.)

- **DIVISION Operation**
  - The division operation is applied to two relations R(Z) ÷ S(X), where X subset Z. Let Y = Z - X (and hence Z = X ∪ Y); that is, let Y be the set of attributes of R that are not attributes of S.
  - The result of DIVISION is a relation T(Y) that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, and with $t_R[X] = t_s$ *for every tuple* t_s in S.
  - For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S.

⋈

THE DIVISION OPERATION. (A) DIVIDING SSN_PNOS BY SMITH_PNOS. (B) $T \leftarrow R \div S$.

(a)

SSN_PNOS	ESSN	PNO
	123456789	1
	123456789	2
	666884444	3
	453453453	1
	453453453	2
	333445555	2
	333445555	3
	333445555	10
	333445555	20
	999887777	30
	999887777	10
	987987987	10
	987987987	30
	987654321	30
	987654321	20
	888665555	20

SMITH_PNOS	PNO
	1
	2

SSNS	SSN
	123456789
	453453453

(b)

R	A	B
	a1	b1
	a2	b1
	a3	b1
	a4	b1
	a1	b2
	a3	b2
	a2	b3
	a3	b3
	a4	b3
	a1	b4
	a2	b4
	a3	b4

S	A
	a1
	a2
	a3

T	B
	b1
	b4

Additional Relational Operations

- **Aggregate Functions and Grouping**
  - A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database.
  - Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples. These functions are used in simple statistical queries that summarize information from the database tuples.
  - Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values.

Additional Relational Operations (cont.)

Use of the Functional operator $\mathcal{F}$

$\mathcal{F}_{\text{MAX } Salary}$ **(Employee)** retrieves the maximum salary value from the Employee relation

$\mathcal{F}_{\text{MIN } Salary}$ **(Employee)** retrieves the minimum Salary value from the Employee relation

$\mathcal{F}_{\text{SUM } Salary}$ **(Employee)** retrieves the sum of the Salary from the Employee relation

$_{\text{DNO}}\mathcal{F}_{COUNT\ SSN,\ AVERAGE\ Salary}$ **(Employee)** groups employees by DNO (department number) and computes the count of employees and average salary per department.[Note: count just counts the number of rows, without removing duplicates]

Additional Relational Operations (cont.)

- **The OUTER JOIN Operation**
  - In NATURAL JOIN tuples without a *matching* (or *related*) tuple are eliminated from the join result. Tuples with null in the join attributes are also eliminated. This amounts to loss of information.
  - A set of operations, called outer joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.
  - The left outer join operation keeps every tuple in the *first* or *left* relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or "padded" with null values.
  - A similar operation, right outer join, keeps every tuple in the *second* or right relation S in the result of R ⟖ S.
  - A third operation, full outer join, denoted by ⟗ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Additional Relational Operations (cont.)

RESULT	FNAME	MINIT	LNAME	DNAME
	John	B	Smith	null
	Franklin	T	Wong	Research
	Alicia	J	Zelaya	null
	Jennifer	S	Wallace	Administration
	Ramesh	K	Narayan	null
	Joyce	A	English	null
	Ahmad	V	Jabbar	null
	James	E	Borg	Headquarters

Additional Relational Operations (cont.)

- **OUTER UNION Operations**
  - The outer union operation was developed to take the union of tuples from two relations if the relations are *not union compatible*.
  - This operation will take the union of tuples in two relations R(X, Y) and S(X, Z) that are **partially compatible**, meaning that only some of their attributes, say X, are union compatible.
  - The attributes that are union compatible are represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation T(X, Y, Z).
  - **Example:** An outer union can be applied to two relations whose schemas are STUDENT(Name, SSN, Department, Advisor) and INSTRUCTOR(Name, SSN, Department, Rank). Tuples from the two relations are matched based on having the same combination of values of the shared attributes—Name, SSN, Department. If a student is also an instructor, both Advisor and Rank will have a value; otherwise, one of these two attributes will be null.

    The result relation STUDENT_OR_INSTRUCTOR will have the following attributes:

    STUDENT_OR_INSTRUCTOR (Name, SSN, Department, Advisor, Rank)

Relational Calculus

- A **relational calculus** expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in **tuple calculus**) or over columns of the stored relations (in **domain calculus**).

- In a calculus expression, there is *no order of operations* to specify how to retrieve the query result—a calculus expression specifies only what information the result should contain. This is the main distinguishing feature between relational algebra and relational calculus.

- Relational calculus is considered to be a **nonprocedural** language. This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence relational algebra can be considered as a **procedural** way of stating a query.

Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

$$\{t \mid P(t)\}$$

- It is the set of all tuples t such that predicate P is true for t
- t is a *tuple variable*, $t[A]$ denotes the value of tuple t on attribute A
- $t \in r$ denotes that tuple t is in relation r
- P is a *formula* similar to that of the predicate calculus

Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or (v), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \text{ v } y$$

5. Set of quantifiers:
   - $\exists\, t \in r\, (Q\,(t\,)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q\,(t\,)$ is true
   - $\forall t \in r\, (Q\,(t\,)) \equiv Q$ is true "for all" tuples t in relation r

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 the *loan_number*, *branch_name*, and *amount* for loans of over $1200

$$\{t \mid t \in loan \wedge t\,[amount\,] > 1200\}$$

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

$$\{t[loan\_number\,] \mid t \in loan \wedge t\,[amount\,] > 1200\}$$

Example Queries

- Find the names of all customers having a loan at the Perryridge branch

$$\{t \mid \exists s \in borrower\ (t\ [customer\_name\ ] = s\ [customer\_name\ ] \wedge \exists u \in loan\ (u\ [branch\_name\ ] = \text{“Perryridge”} \wedge u\ [loan\_number\ ] = s\ [loan\_number\ ]))\}$$

OR

$$\{t[customer\_name] \mid t \in borrower \wedge t\ [branch\_name\ ] = \text{“Perryridge”} \wedge \exists s \in loan\ (s\ [loan\_number\ ] = t\ [loan\_number\ ])\}$$

Example Queries

- Find the names of all customers having a loan, an account, or both at the bank

$$\{t \mid \exists s \in borrower\ (\ t\ [customer\_name\ ] = s\ [customer\_name\ ])$$
$$\vee \exists u \in depositor\ (\ t\ [customer\_name\ ] = u\ [customer\_name\ ])$$

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

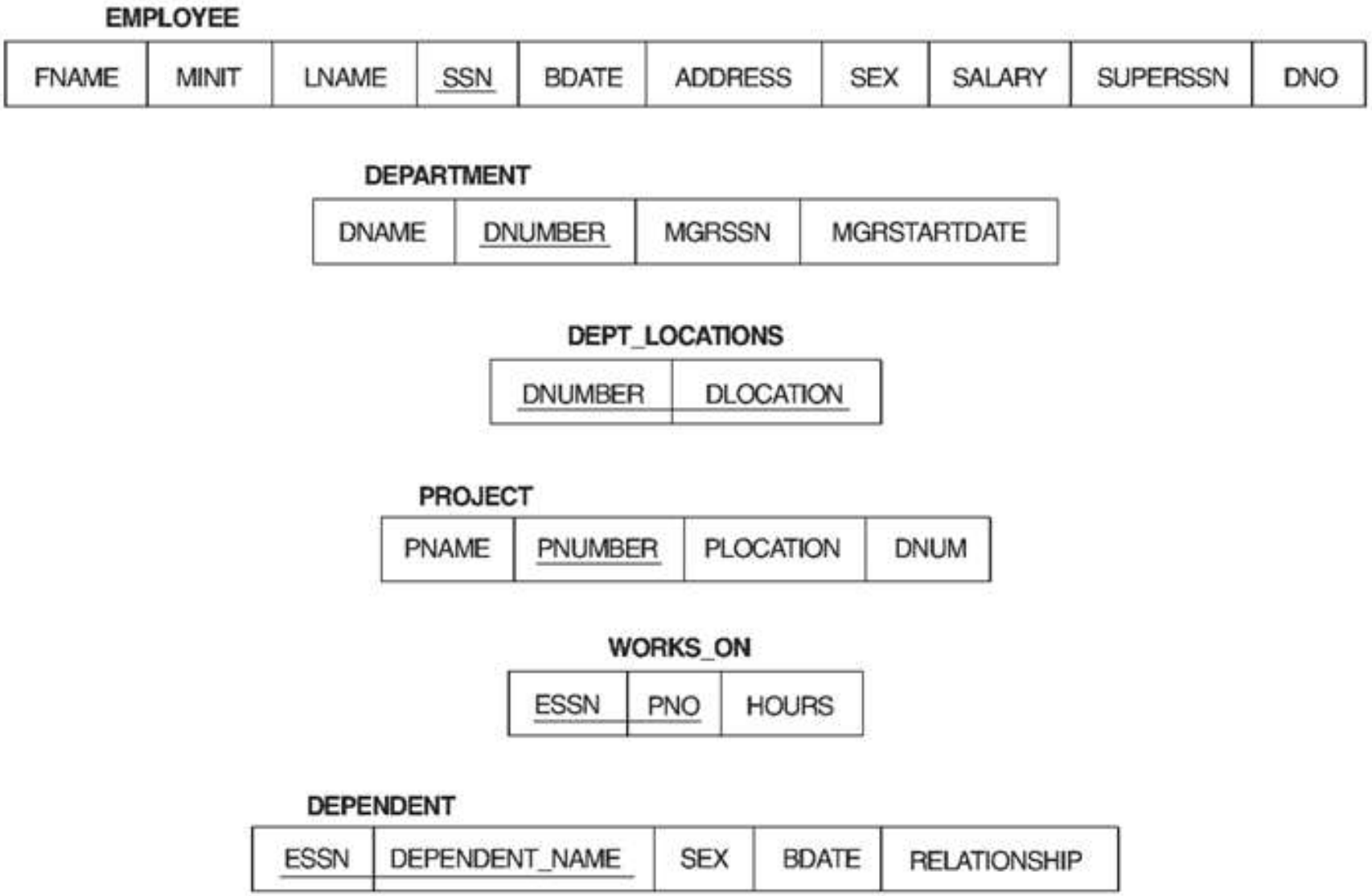
$$\{t \mid \exists s \in borrower\ (\ t\ [customer\_name\ ] = s\ [customer\_name\ ])$$
$$\wedge \exists u \in depositor\ (\ t\ [customer\_name\ ] = u\ [customer\_name])$$

The Existential and Universal Quantifiers

- Two special symbols called **quantifiers** can appear in formulas; these are the **universal quantifier** (∀) and the **existential quantifier** (∃).
- Informally, a tuple variable t is bound if it is quantified, meaning that it appears in an (∀ t) or (∃ t) clause; otherwise, it is **free.**
- If F is a formula, then so is (∃ t)(F), where t is a tuple variable. The formula (∃ t)(F) is true if the formula F evaluates to true for *some* (at least one) tuple assigned to free occurrences of t in F; otherwise (∃ t)(F) is **false.**
- If F is a formula, then so is (∀ t)(F), where t is a tuple variable. The formula (∀ t)(F) is true if the formula F evaluates to true for *every tuple* (in the universe) assigned to free occurrences of t in F; otherwise (∀ t)(F) is **false.**

  It is called the universal or "for all" quantifier because every tuple in "the universe of" tuples must make F true to make the quantified formula true.

Schema diagram for the COMPANY relational database schema; the primary keys are underlined.



Example Query Using Existential Quantifier

- Retrieve the name and address of all employees who work for the 'Research' department.

  Query :

  {t.FNAME, t.LNAME, t.ADDRESS | EMPLOYEE(t) and (∃ d) (DEPARTMENT(d) and d.DNAME='Research' and d.DNUMBER=t.DNO) }

- The *only free tuple variables* in a relational calculus expression should be those that appear to the left of the bar (|). In above query, t is the only free variable; it is then *bound successively* to each tuple. If a tuple *satisfies the conditions* specified in the query, the attributes FNAME, LNAME, and ADDRESS are retrieved for each such tuple.

- The conditions EMPLOYEE (t) and DEPARTMENT(d) specify the range relations for t and d. The condition d.DNAME = 'Research' is a selection condition and corresponds to a SELECT operation in the relational algebra, whereas the condition d.DNUMBER = t.DNO is a JOIN condition.

Example Query Using Universal Quantifier

- Find the names of employees who work on *all* the projects controlled by department number 5.

  Query :

  {e.LNAME, e.FNAME | EMPLOYEE(e) and ((∀ x)((PROJECT(x)) and x.DNUM=5 OR ((∃ w)(WORKS_ON(w) and w.ESSN=e.SSN and x.PNUMBER=w.PNO))))}

- Exclude from the universal quantification all tuples that we are not interested in by making the condition true *for all such tuples*. The first tuples to exclude (by making them evaluate automatically to true) are those that are not in the relation R of interest.
- In query above, using the expression not(PROJECT(x)) inside the universally quantified formula evaluates to true all tuples x that are not in the PROJECT relation. Then we exclude the tuples we are not interested in from R itself. The expression not(x.DNUM=5) evaluates to true all tuples x that are in the project relation but are not controlled by department 5.

- Finally, we specify a condition that must hold on all the remaining tuples in R.

  ((∃w)(WORKS_ON(w) and w.ESSN = e.SSN and x.PNUMBER = w.PNO)

Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, { t | $\neg$ $t \in r$ } results i an infinite relation if the domain of any attribute of relation *r* is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression {*t* | *P* (*t*)} in the tuple relational calculus is *safe* if every component of *t* appears in one of the relations, tuples, or constants that appear in *P*.
  - *Ex:* {t | $\neg$ t[name] = 'X'} will display the tuples from the whole database having name not equal to 'X'.
  - *Safe Expression:*

    {t | $\neg$ t[name] = 'X' $\wedge$ t $\in$ emp}

The Domain Relational Calculus

- Another variation of relational calculus called the domain relational calculus, or simply, **domain calculus** is equivalent to tuple calculus and to relational algebra.

- The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York. Domain calculus was thought of as a way to explain what QBE does.

- Domain calculus differs from tuple calculus in the *type of variables* used in formulas: rather than having variables range over tuples, the variables range over single values from domains of attributes. To form a relation of degree n for a query result, we must have n of these **domain variables**—one for each attribute.

- An expression of the domain calculus is of the form

  $\{x1, x2, \ldots, xn \mid COND(x1, x2, \ldots, xn, xn+1, xn+2, \ldots, xn+m)\}$

  where x1, x2, . . ., xn, xn+1, xn+2, . . ., xn+m are domain variables that range over domains (of attributes) and COND is a **condition** or **formula** of the domain relational calculus.

Example Query Using Domain Calculus

- Retrieve the birthdate and address of the employee whose name is ‘John B. Smith’.

  Query :

  {uv | (∃ q) (∃ r) (∃ s) (∃ t) (∃ w) (∃ x) (∃ y) (∃ z)
  (EMPLOYEE(qrstuvwxyz) and q=’John’ and r=’B’ and s=’Smith’)}

- Ten variables for the employee relation are needed, one to range over the domain of each attribute in order. Of the ten variables q, r, s, . . ., z, only u and v are free.
- Specify the *requested attributes*, BDATE and ADDRESS, by the free domain variables u for BDATE and v for ADDRESS.
- Specify the condition for selecting a tuple following the bar (|)—namely, that the sequence of values assigned to the variables qrstuvwxyz be a tuple of the employee relation and that the values for q (FNAME), r (MINIT), and s (LNAME) be ‘John’, ‘B’, and ‘Smith’, respectively.

Example Queries

- Find the *loan_number*, *branch_name*, and *amount* for loans of over $1200

$$\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$$

- Find the names of all customers who have a loan of over $1200

$$\{< c > \mid \exists\, l, b, a\, (< c, l > \in borrower \wedge < l, b, a > \in loan \wedge a > 1200)\}$$

- Find the names of all customers who have a loan from the Perryridge branch and the loan amount:
  - $\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge \exists b\, (< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$

Safety of Expressions

The expression:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n) \}$$

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from *dom* (P) (that is, the values appear either in P or in a tuple of a relation mentioned in P).
2. For every "there exists" subformula of the form $\exists x (P_1(x))$, the subformula is true if and only if there is a value of x in *dom* (P_1) such that $P_1(x)$ is true.
3. For every "for all" subformula of the form $\forall_x (P_1(x))$, the subformula is true if and only if $P_1(x)$ is true for all values x from *dom* (P_1).