

3. RELATIONAL MODEL

This example database for a tiny credit card company will be used frequently in this section of the course.

v (Vendor)

Vno	Vname	City	Vbalance
V1	Sears	Toronto	200.00
V2	WalMart	Waterloo	671.05
V3	Esso	Windsor	0.00
V4	Esso	Waterloo	225.00

c (Customer)

Account	Cname	Province	Cbalance	Crlimit
A1	Smith	ONT	2515.00	2000
A2	Jones	BC	2014.00	2500
A3	Doc	ONT	150.00	1000

t (Transaction)

Tno	Vno	Account	T_Date	Amount
T1	V2	A1	020915	1325.00
T2	V2	A3	010916	1900.00
T3	V3	A1	020915	2500.00
T4	V4	A2	010920	1613.00
T5	V4	A3	000925	3312.00

3.1 Basic Structure

- In a relational database, data are stored in tables.
- A **table** stores data about a type of entities or relationships.
 - A **column** is an attribute.
 - For each attribute, there is a set of permitted values, called the **domain** of that attribute.
 - A **row** represents an instance of entity or relationship. It is an ordered set of values, one for each attribute.
- A **relation** is a table.
- Mathematically, a relation is defined to be a subset of a Cartesian product of a list of domains.
- Representing data as relations allows us to use methods in set theory to manipulate data.

3.2 Basic Concepts

- An **attribute** is a column in a relation/table.
- A **null value** is a special value meaning “not known”, “not applicable”, “no information”, ...
- A **relation schema** defines the structure of a relation by specifying its name, its attributes and their types.

- A **tuple** is a row in a relation. It is an (ordered) set of values, one for each attribute in the relation schema over which the tuple is defined.
- A **relation (instance)** is a set of tuples over the schema of the relation. In other words, it is a table with one column for each attribute in the schema.
- A **relational database** is a set of relations, each one with a unique name.
- In the same way as for relations, we distinguish between the schema and an instance of the database.
 - A **relational database schema** is a set of relation schemas.
 - A **relational database** is a set of relations.
- In a **normalized relation** every value in the relation is atomic, that is, every attribute in every tuple contains a single value from an atomic (or primitive) data type, not a set of values.
- We will use R for relation schemas and r for relation instances.
- Keys:
 - A **superkey** of a relation is a set of one or more attributes which, taken collectively, allow us to identify a tuple in the relation uniquely. No two tuples can have the same values for this set of attributes.
 - A **candidate key** is a minimal set of attributes which has the uniqueness property (i.e. which is a superkey).
 - The **primary key** of a relation is a candidate key designated as the main key of the relation.

- A **foreign key** of a relation is an attribute (or combination of attributes) whose values are restricted to a subset of the values of the primary key of another relation (the referenced relation). A foreign key is also called a connection key or reference attribute.

Example:

Dept (**dept-id**, name, location)

Prof (prof-id, name, **dept-id**)

- An **integrity constraint** is a condition that must be satisfied by every valid instance of a database/table/column.

Integrity rules

- Entity integrity

Primary key values must be unique and no component of a primary key value may be null. Motivation: a tuple must represent some unique, identifiable entity in the real world.

- Referential integrity

A tuple with a value for a foreign key that does not match the primary key value of a tuple in the referenced relation is not allowed. In other words, a foreign key must reference a tuple that exists.

3.3 Relational Algebra

Relational algebra is a theoretical query language for relational databases. It has a set of operators which operate on relations and produce new relations.

- **Set Operations**

– Union: $r_1 \cup r_2$

The union of r_1 and r_2 contains the tuples that are in either r_1 or r_2 .

– Intersection: $r_1 \cap r_2$

The intersection of r_1 and r_2 contains the tuples that are in both r_1 and r_2 .

– Set Difference: $r_1 - r_2$

The difference between r_1 and r_2 contains the tuples that are in r_1 but not in r_2 .

– **Note:** r_1 and r_2 must be (union-) compatible for the above 3 operations, that is, have the same set of attributes (the same schema).

– Cartesian Product: $r_1 \times r_2$

The Cartesian product contains the tuples formed by concatenating a tuple in r_1 with a tuple in r_2 , for every possible combination of a tuple in r_1 and a tuple in r_2 . (This may result in a *very* large relation.)

Example: (Cartesian product)

r_1

r_2

$r_1 \times r_2$

• Relational operators

– Selection: $\sigma_C(r)$

where C is a selection condition. Let R be the schema of r . C is specified in terms of the attributes in R . A selection operation selects all tuples from a relation that satisfy a given condition, that is, extracts a “horizontal” subset of the relation.

A	B
1	x
2	y

C	D
a	s
b	t
c	u

A	B	C	D
1	x	a	s
1	x	b	t
1	x	c	u
2	y	a	s
2	y	b	t
2	y	c	u

Example: Find all vendors in Waterloo.

$$\sigma_{\text{City} = \text{“Waterloo”}}(v)$$

Vno	Vname	City	Vbalance
V2	WalMart	Waterloo	671.05
V4	Esso	Waterloo	225.00

– Project: $\Pi_A(r)$

where A is a list of attributes of r . That is, $A \subseteq R$.

A projection operation constructs a new relation by deleting all attributes (columns) from a relation which are not mentioned in A , and eliminating duplicate tuples. In other words, projection extracts a “vertical” subset of the relation.

Example:

List all cities where there is a vendor.

$\Pi_{\text{City}}(v)$

City
Toronto
Waterloo
Windsor

– Join (natural): $r_1 \bowtie r_2$

Assume that the attributes common (with the same name) to R_1 and R_2 are A_1, \dots, A_m . Then the natural join constructs a new relation by (pairwise) concatenation of all tuples from r_1 and r_2 that satisfy the condition $r_1.A_1 = r_2.A_1, \dots, r_1.A_m = r_2.A_m$, and eliminating attributes $r_2.A_1, \dots, r_2.A_m$ from the result.

$$r_1 \bowtie r_2 = \Pi_A(\sigma_C(r_1 \times r_2))$$

where $A = \{\text{all attributes of } R_1 \text{ and } R_2$

except $R_2.A_1, \dots, R_2.A_m\}$ and

$$C = (r_1.A_1 = r_2.A_1, \dots, r_1.A_m = r_2.A_m)$$

Example:

Join vendor information with transactions. $t \bowtie v$

– Generalized join (theta-join):

$$r_1 \bowtie_{\theta} r_2 = \sigma_{\theta}(r_1 \times r_2)$$

where θ can be any boolean expression formed by using $=, \neq, <, \leq, >, \geq, \vee, \wedge, \neg$, and the attributes of R_1 and R_2 .

Tno	Vno	Account	...	Vname	City	Vbalance
T1	V2	A1	...	WalMart	Waterloo	671.05
T2	V2	A3	...	WalMart	Waterloo	671.05
T3	V3	A1	...	Esso	Windsor	0.00
T4	V4	A2	...	Esso	Waterloo	225.00
T5	V4	A3	...	Esso	Waterloo	225.00

– Outer join: $r_1 \bowtie^{right} R_2$

Same as a natural join with the exception that all tuples in r_2 (the one indicated by the superscript *right*) are retained.

If a tuple of r_2 does not (naturally) join with any tuple in r_1 , it is extended with null values for all attributes of r_1 and the resulting tuple is output.

Example:

Join vendors and transactions, retaining all vendors.

$t \bowtie^{right} v$

Tno	Vno	Account	...	Vname	City	Vbalance
null	V1	null	...	Sears	Toronto	200.00
T1	V2	A1	...	WalMart	Waterloo	671.05
T2	V2	A3	...	WalMart	Waterloo	671.05
T3	V3	A1	...	Esso	Windsor	0.00
T4	V4	A2	...	Esso	Waterloo	225.00
T5	V4	A3	...	Esso	Waterloo	225.00

The outer join discussed above is a **right outer join**. We may also have **left outer join** and **full outer join**, which can be expressed using \bowtie^{left} and \bowtie^{full} .

- Division: $r_1 \div r_2$

Assume that R_1 and R_2 are the schemas of r_1 and r_2 , the attributes of R_2 is a subset of those of R_1 , and R_1 has m additional attributes. The relation of $r_1 \div r_2$ is on the m attributes.

$$r_1 \div r_2 = \Pi_{R_1 - R_2}(r_1) - \Pi_{R_1 - R_2}((\Pi_{R_1 - R_2}(r_1) \times r_2) - \Pi_{R_1 - R_2, R_2}(r_1))$$

3.4 Transforming E-R to Relational

Transforming entity sets

- Each entity set is transformed into a relation that includes all the attributes of the entity set.
- The primary key of the entity set becomes the primary key of the corresponding relation.
- Each nonkey attribute of the entity set becomes a nonkey attribute of the relation.

Transforming relationship sets

- A **binary 1:N relationship set** can be represented by adding the primary key of the entity set on the one-side of the relationship set, as a foreign key in the relation that corresponds to the entity set on the many-side of the relationship set.

- A **binary M:N relationship set** can be represented by creating a separate relation. The primary key of the new relation is a composite key consists of the primary keys of the entity sets in the relationship set. Any nonkey attributes associated with the relationship set are included in the new relation.
- In a **unary 1:N relationship set**, the entity set is transformed into a relation. The primary key of the relation is the same as for the entity set. Then a foreign key is added to the relation that references the primary key values.
- For a **unary M:N relationship set**, the entity set is transformed into a relation. Then a separate relation is created to represent the M:N relationship set. The primary key of the new relation is a composite key of two attributes which take values from the same primary key domain.
- An **n-ary relationship set** can be represented by creating a separate relation. The primary key of the new relation is a composite key consists of the primary keys of the entity sets in the relationship set. Any nonkey attributes associated with the relationship are included in the new relation.
- For an **ISA relationship**,
 - create a separate relation for the super-set and each of the sub-sets,
 - include the common attributes in the relation for the super-set,
 - include the primary key (of the super-set) and the attributes unique to a sub-set in the relation for the sub-set.

3.5 Relational Calculus

- **Note:** Syntax may be different from those in the reference books.
- Relational calculus is nonprocedural; a relational calculus expression is a formal definition of a new relation in terms of other relations.
- Two variants of relational calculus: tuple calculus and domain calculus. (Domain calculus not considered in this course.)
- **Tuple variable** = a variable that ranges over a named relation, that is, whose values are all tuples from that relation. It can also be considered as a pointer moving throughout all the tuples in the relation.
- Example 1:

Get the names of all vendors in Waterloo.

$$\forall vx(v) \\ \{ \langle vx.Vname \rangle : vx.City='Waterloo' \}$$

- **Syntax:**

$$\forall t_1(r_1), \forall t_2(r_2), \dots, \forall t_n(r_n) \\ \{ \langle \text{target-list} \rangle : \text{predicate} \}$$

where t_1, t_2, \dots, t_n are tuple variables, r_1, r_2, \dots, r_n are relation names, “target-list” specifies the attributes of the resulting relation, and “predicate” is a boolean formula giving a condition that tuples must satisfy to qualify for the resulting relation.

- The predicate is constructed from
 - attribute names

- constants
- comparison operators ($=, \neq, >, \geq, <, \leq$)
- logical connectives (\vee, \wedge, \neg)
- quantified tuple variables ($\forall t(r), \exists t(r)$)

- **Semantics:**

Every tuple in the Cartesian product of t_1, t_2, \dots, t_n that satisfies the predicate is part of the result. The final result is obtained after projection onto the attributes in target-list and elimination of duplicates.

The query in **Syntax** is equivalent to

$$\Pi_{\text{target-list}}(\sigma_{\text{predicate}}(r_1 \times r_2 \times \dots \times r_n))$$

- Example 2:

Get the dates, amounts and vendor names of all transactions by customer A2.

$$\begin{aligned} &\forall \text{tx}(t), \forall \text{vx}(v) \\ &\{ \langle \text{tx.T_Date}, \text{tx.Amount}, \text{vx.Vname} \rangle : \\ &\quad \text{tx.Account} = \text{'A2'} \wedge \text{tx.Vno} = \text{vx.Vno} \} \end{aligned}$$

- Example 3:

Get the account numbers and names of the customers having at least one transaction.

$$\begin{aligned} &\forall \text{cx}(c) \\ &\{ \langle \text{cx.Account}, \text{cx.Cname} \rangle : \\ &\quad \exists \text{tx}(t)(\text{tx.Account} = \text{cx.Account}) \} \end{aligned}$$

Any tuple variable not “declared” in the first part of the query must be *bound* in the predicate. A tuple variable in a predicate is *bound* if it is quantified in the predicate, otherwise it is *free* (in the predicate).

In the example “tx” is bound and “cx” is free in the predicate.

Every tuple variable in the target list must be universally quantified.

- Example 4:

Get the account numbers and names of the customers whose transactions are all with vendor V4.

$$\begin{aligned} & \forall cx(c) \\ & \{ \langle cx.Account, cx.Cname \rangle : \\ & \quad \forall tx(t)(tx.Account=cx.Account \Rightarrow tx.Vno='V4') \\ & \quad \wedge \exists ty(t)(ty.Account=cx.Account) \} \end{aligned}$$

Note that the implication is also true for customers not having any transactions. The second part of the predicate is necessary to disqualify such customers.

- Example 5:

Get the account numbers and names of all customers who have transactions with both vendors V1 and V2.

$$\begin{aligned} & \forall cx(c) \\ & \{ \langle cx.Account, cx.Cname \rangle : \\ & \quad \exists tx1(t)(cx.Account=tx1.Account \wedge tx1.Vno='V1') \\ & \quad \wedge \exists tx2(t)(cx.Account=tx2.Account \wedge tx2.Vno='V2') \} \end{aligned}$$

- Example 6:

Get the account numbers and names of all customers who have no transactions with Esso.

$$\begin{aligned} & \forall cx(c) \\ & \{ \langle cx.Account, cx.Cname \rangle : \\ & \quad \neg (\exists tx(t), \exists vx(v) (cx.Account=tx.Account \\ & \quad \wedge tx.Vno=vx.Vno \wedge vx.Vname='Esso')) \} \end{aligned}$$

Note that customers with no transactions at all will also qualify.

- Example 7:

Get vendor numbers and names of the vendors who have transactions with all customers in Ontario.

$$\begin{aligned} & \forall vx(v) \\ & \{ \langle vx.Vno, vx.Vname \rangle : \\ & \quad \forall cx(c) (cx.Province='ONT' \Rightarrow \\ & \quad \exists tx(t) (tx.Account=cx.Account \wedge tx.Vno=vx.Vno)) \} \end{aligned}$$

Reading: Sections 3.1, 3.2, 5.1, 5.2