The Relational Model

- Structure of Relational Databases
  - **Mathematical Relations**
  - **Relational Schema & Relational Instances**
  - **Keys**
- Relational Algebra
  - **Relational Algebra Operations**
  - **Example queries**

The Relational Model

- **Proposed by E.F.Codd in the early seventies.**
- **Most of the modern DBMS are relational.**
- **Simple and elegant model with mathematical basis.**
- **Led to the development of a theory of data dependencies and database design.**
- **Relational algebra operations –crucial role in query optimization & execution.**
- **Laid the foundation for the development of**
  - **Tuple relational calculus and then**
  - **Database standard SQL**

Mathematical Relations

- The relational model is built upon the concept of Mathematical Relations
- A relation is a (not necessarily proper) subset of a Cartesian product on n sets(Domains).

  D1 x D 2x …. x Dn = { (d 1, d2,….dn) / d1 ϵ D1 , d2 ϵ D2,… dn ϵ Dn }

  D1,D2…Dn are domains of the elements in the relation.

  E.g. {(1, a), (1, e) ,(2, e)} ≤ {1, 2} x{ a,c,e}

  It is a relation on the sets {1, 2, g} and { a, c, e}

Relation Schema

- *The mathematical formality can be used in the database context to define relation schemas .*
- *A relation schema* consists of relation name,and a set of attributes or field names or column names.Each attribute has an associated domain.Domain is the set of allowed values of the attribute.

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

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

Attribute Names *Domains*

Customer (customer-Id : Digits(10),

customer-name *:Char(60),*

Relation Name *customer-street* *: Char(30),*

Category *: { A,B,C}*)

- *When the domains are known, shorthand notation can be used:*

Customer (customer-Id, *customer-name, customer-street, customer-city*)

Relation Instance

- An element of a relation is called a *tuple.*
- A relation instance is the collection of tuples of a relation at a particular moment in time. R*elation instance* of a relation are specified by a table. So tuples are rows in the table.
- Degree of a relation : The number of attributes (e.g. ---4)
- Cardinality of a relation : The number of tuples (e.g.---4)

customer

attributes (or columns)

customer-ID	*customer-name*	*customer-street*	Category
5700893	Jones	Main	A
5700379	Smith	NULL	A
5700321	Curry	North	C
5700990	Lindsay	NULL	B

tuples (or rows)

Attribute Types

- Attribute values are (normally) required to be **atomic**, that is, indivisible
  - ***Multi valued* attribute values are not atomic (e.g. phone-number)**
  - ***composite* attribute values are not atomic (e.g. address)**
- The special value *null* is a member of every domain.It represent the unknown values or unspecified values.

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- The following two tables represent the *same* relation:

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

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

- No significance to ordering
- because both tables represent sets

Relational Database Schema

- A database consists of one or more relations
- Information about an enterprise is broken down into its components
- Each relation stores just part of the information

  E.g.: *account* : information about accounts
  depositor : information about which customer owns which account
  customer : 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. two customers own an account)**
  - **the need for null values (e.g. represent a customer without an account)**

Relational Database Schema (Cont.)

- A relational database schema is a set of relation schemas, each with a distinct name.

E.g.: Relational database schema { *account , customer }*

account(Account No: Digits(10),
Balance : Digits(8)) and
customer (Customer-name : Char (60),
Account No : Digits(10)) are the two relations in the database schema..

Keys

- Retrieval of data from a relation requires keys
- There are dierent types of keys:
- Super key
- Candidate key
- Primary key and
- Foreign key

Super keys

- A set of attributes K, whose values uniquely identify a tuple in any instance.
- If K is a super key any super set of k is also a super key.

customer

customer-ID	*customer-name*	*customer-street*	Category
5700893	Jones	Main	A
5700379	Smith	NULL	A
5700321	Curry	North	C
5700990	Lindsay	NULL	B

{customer-ID},{customer-ID ,category},
***{customer-name,customer-street}*, ---(if no two customers have the same name)**
(customer-name},
{customer-ID} etc. are superkeys

Candidate Keys

- *They are super keys for which no proper subset is a super key.*
- *i.e. K* is a ***candidate key*** if:
  - ***K* is a super key**
  - ***K* is minimal (contains no super keys)**

{*customer-name*}, { customer-ID} are candidate keys for *Customer*

Primary Key

- **A primary key is a candidate key chosen by the database designer to identify tuples uniquely within a relation**
- **Primary keys are often underlined in a relation schema**
- **Note that a primary key is a candidate key, and a candidate key is a super key**
- **Candidate keys that are not the primary key are called alternate keys**

e.g. Customer (<u>customer-ID</u>, customer-name, customer-street, category)

Foreign Keys

- Relation schema may contain attributes which refers to a candidate key attributes of some other relation provided both the attribute and the candidate key attributes are having the same domain. Such keys are called foreign keys.
- Example: Employee (<u>EmpId</u> , EmpName,DepNo)

  Department (<u>DepId</u>, DepName)

  The foreign key attribute DepNo in Employee relation refers to the primary key attribute DepId in Department relation.
- There can be more than one foreign key in a relation schema.
- It is possible for a foreign key in a relation to refer to the primary key of the relation itself

  Example: univEmployee (empNo , name, sex, salary, dept, reportsTo)
  reportsTo is a foreign key referring to *empNo* of the same relation.

  Every employee in the university reports to some other employee for administrative purposes - except the *vice- chancello*r, of course!

Relational Algebra

- A formal query language associated with the relational model.
  Gives a procedural method of specifying a retrieval query.
- Consist of functions that accept relation instances as arguments and return a relation instance as result
- Forms the core component of a relational query engine.
- SQL queries are internally translated into RA expressions.
- RA provides a framework for query optimization.
- There are five fundamental operations, two auxiliary operations, and a number of operations derived from them.

Relational Algebra Fundamental Operations

Let R and S be expressions resulting in relation instances

- Selection $\sigma_\theta(R)$ ------------ choose tuples (rows)
- Projection $\pi_{attrList}(R)$ ----------- choose attributes columns
- Union $R \cup S$ ------------- pool tuples together
- Set difference $R - S$ ------------- tuples in R but not in S
- Cartesian product $R \times S$ ------------ all tuple combinations

Relational Algebra Auxiliary Operations

These are used mainly for resolving naming conflicts and to break down complex operations to simpler ones

- Assignment $R \leftarrow S$ ----------set the value of R to that of S
- Rename $\rho_{newName}(R)$ ----------change attribute names

Relational Algebra Other Operations

- Set intersection $R \cap S$ ---------- tuples in both R and S
- Conditional join $R \bowtie_{\theta} S$ ---------- selection on R X S
- Equijoin --conditional join with only equality comparison in predicate
- Natural join $R \bowtie S$ ---------- Equijoin on common attributes
- Division $R \div S$ ----------- "inverse" of R X S

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

Selection

- A unary operation : $\sigma_{\theta}(R)$
- Selects tuples (rows) from a relation instance
- Resultant relation instance has the same schema as R
- It contains only tuples of R that satisfies the predicate θ
- The predicate is a Boolean expression. It consists of a combination of terms are comparison between two attribute values or between an attribute value and a constant.
- Allowed comparison are $<, \leq, >, \geq, =, \neq$ Terms are connected by logical and ($\wedge$) or logical or ($\vee$) operators

Selection – 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

Projection

- A unary operation: $\pi_{attrList}$ (R) Project attributes (columns) from a relation instance
- Resultant relation schema consist of attributes in attrList in the specified order
- Each tuple in the answer comes from a tuple in R, with only the specified attributes retained
- Duplicate tuples are eliminated

Projection-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

- A binary operation: R ∪ S
- Resultant relation instance has schema the same as that of R (or S)
- R ∪ S is the set union of all the tuples that occur in either, or both, relation instances R or S
- R and S must be (union-)compatible. Compatible relations must satisfy the following criteria: they have the same number of attributes corresponding attributes, taken in order from left to right, have the same domains
- There are no duplicate tuples in the resultant relation instance

Union – 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*
α	*2*
β	*1*
β	*3*

Set Difference

- A binary operation: R – S
- Resultant relation instance has schema the same as that of R (or S)
- The result contain tuples that occur in R but not in S .
- R and S must be compatible

Set Difference – 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

- A binary operation: R x S
- Also called cross-product and Similar to Cartesian product of sets
- Resultant relation instance has schema that contains, in order, all the attributes in R and then all the attributes in S. For every tuple r ϵ R and s ϵ S, the concatenation of r and s is a result tuple of R x 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.

Cartesian-Product -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

Assignment

- Binary operation: R ←S
- Content of R is replaced by the content of S
- R and S must be compatible
- Resultant relation instance has the schema of R

Example: T ← R x S:

Rename

- Gives relations or attributes a different name.
- 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. Rename gives relations or attributes a different name
- Unary operation
- Has a number of variations
- $\rho_X(R)$ ------ rename the relation instance R as X
- $\rho_{X(A1,A2,\ldots An)}(R)$ -------- rename the relation instance R with degree n as X, and the r-th attribute as Ar for r ϵ { 1,.. n }
- $\rho_{(X1\rightarrow Y1, X2\rightarrow Y2,\ldots Xm\rightarrow Ym)}(R)$ ----- rename attribute name x_r as y_r in the relation instance R for r ϵ {1,2,…m}
- $\rho x_{(X1\rightarrow Y1, X2\rightarrow Y2,\ldots Xm\rightarrow Ym)}(R)$ rename attribute name x_r as y_r for r ϵ {1,2,.. m } in the relation instance R renamed as X

Additional Operations

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

- Set intersection
- Join Operations
- Division
- Assignment

Set intersection

- A binary operation: $R \cap S$
- The result contain tuples that occur both in R and S
- Resultant relation instance has schema the same as that of R (or S)
- R and S must be compatible
- $R \cap S = R - (R - S)$

Set-Intersection - 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

Join operations

- Combine information from two or more relations
- Selections and projections of cross product of two relation instances
- A number of variants:
  - **Conditional join (θ-join)**
  - **Equijoin**
  - **Natural join**

Conditional join (θ-join)

- $R \bowtie_{\theta} S$ where R and S are relations and is θ is a predicate on the attributes of R together with that of S
- The predicate often refer to attributes of both R and S
- $R \bowtie_{\theta} S = \sigma_{\theta} (R \times S)$

θ- Join Example

- Relations R, S:

A	B
α	1
β	2
γ	4
α	1
δ	2

R

C	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

S

$R \bowtie_{B > C} S$

A	B	C	D	E
γ	4	1	a	α
γ	4	3	a	β
γ	4	1	a	γ
γ	4	2	b	δ
γ	4	3	b	$\in$

Equijoin

- Equijoin is a special case of conditional join
- Predicate of a conditional join may contain only conjunction (logical AND) of equalities between two attributes in the two relations
- For $R \bowtie_{R.a=S.b} S$, it is redundant to include both R.a and S.b in the result
- A projection is done so that only the R.a is retained
- Join operation with this refinement is called equijoin

Equi Join Example

- Relations R, S:

A	B
α	1
β	2
γ	4
δ	2

R

C	D	E
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

S

$R \bowtie_{B = C} S$

A	B	D	E
α	1	a	α
β	2	b	δ
δ	2	b	δ

Natural-Join

- Notation: $R \bowtie S$
- Equality on attributes with the same name
- Equijoin in which equalities are specified on all fields having the same name in the operands
- The result will not have two fields with the same name

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	$\in$

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

Outer Joins

- Theta join,equi-join and natural join are all called *inner joins.* The result of these operations contain only the matching tuples.
- The set of operations called *outer joins* are used when all tuples in relation R or relation S or both in R and *S* have to be in result.
- There are 3 kinds of outer joins:
  - **Left outer join** **R ⟕ S**
  - **Right outer join** **R ⟖ S**
  - **Full outer join** **R ⟗ S**

Outer Joins (Cont.)

- Left outer join: It keeps all tuples in the first, or left relation R in the result. For some tuple *t* in R, if no matching tuple is found in *s* then S- attributes of *t* are made null in the result.
- Right outer join: Same as above but tuples in the second relation are all kept in the result. If necessary, R- attributes are made null.
- Full outer join: All the tuples in both the relations *r* and *s* are in the result.

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

Inner Join

- **Inner Join**

loan ⋈ *Borrower*

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

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

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

Left Outer Join

- **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*

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

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

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

Division Operation

- Binary operation: R ÷ S, or R / S
- Resultant relation schema is that of R without attributes in S (i.e R – S)
- A tuple will appear in the result if every combination tuples in S and itself appears in R
- Suited to queries that include the phrase "for all".
- $R\ (A_1, \ldots, A_m, B_1, \ldots, B_n)$

  $S(B_1, \ldots, B_n)$

  The result of R ÷ S is a relation on schema

  $\boldsymbol{R - S\ (A_1, \ldots, A_m)}$

  R ÷ S =

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 ÷ 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 ÷ S:

A	*B*	*C*
α	a	γ
γ	a	γ

Why “division"

- Integer division: let Q = A / B, where Q, A, and B are all integers. Q is the greatest integer such that Q x B <= A
- Relational algebra division: let Q = A ÷ B, where Q, A,and B are all relation instances. Loosely, Q is the largest relation instance such that Q X B $\subseteq$ A
- Practical systems often do not implement division

Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

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 Rs.1200

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

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

$$\Pi_{loan\text{-}number}\ (\sigma_{amount > 1200}\ (loan))$$

Example Queries

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

$$\Pi_{customer\text{-}name}\ (borrower) \cup \Pi_{customer\text{-}name}\ (depositor)$$

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

$$\Pi_{customer\text{-}name}\ (borrower) \cap \Pi_{customer\text{-}name}\ (depositor)$$

Example Queries

- Find the names of all customers who have a loan at the Idukki branch.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Idukki”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$$

- Find the names of all customers who have a loan at the Idikki branch but do not have an account at any branch of the bank.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name\ =\ \text{“Idukki”}}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(\text{borrower x loan})))\ -\ \Pi_{customer\text{-}name}(\text{depositor})$$

Example Queries

- Find the names of all customers who have a loan at the Idukki branch.
  - Query 1

$$\Pi_{\text{customer-name}}(\sigma_{\text{branch-name = “Idukki”}}\ (\sigma_{\text{borrower.loan-number = loan.loan-number}}(\text{borrower x loan})))$$

  - Query 2

$$\Pi_{\text{customer-name}}(\sigma_{\text{loan.loan-number = borrower.loan-number}}((\sigma_{\text{branch-name = “Idukki”}}(\text{loan}))\ \text{x}\ \ \text{borrower}))$$

Example Queries

Find the largest account balance

- Rename *account* relation as *d*
- The query is:

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

Example Queries

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

Query 1

$$\Pi_{CN}(\sigma_{BN=\text{"Thodupuzha"}}(depositor \bowtie account)) \cap$$
$$\Pi_{CN}(\sigma_{BN=\text{"Idukki"}}(depositor \bowtie account))$$

where *CN* denotes customer-name and *BN* denotes *branch-name*.

Example Queries

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

$$\Pi_{customer\text{-}name,\ branch\text{-}name}\ (depositor \bowtie account)$$
$$\div\ \Pi_{branch\text{-}name}\ (\sigma_{branch\text{-}city\ =\ \text{"Kochi"}}\ (branch))$$