Relational Model

- Most popular model for the implementation of databases
- Does not get tied to the underlying implementation of database
- Substantial theoretical developments exist

Structure of Relational Databases

- Collection of *tables*, each with a unique name
- Each row in a table represents a *relationship* among a set of values
- Basic Structure

Relation: `Supplier`			
`S_Num`	`S_Name`	`Status`	City
s_1	Smith	20	London
s_2	Jones	10	Paris
s_3	Blake	30	Paris
s_4	Clark	20	London
s_5	Adams	30	Athens

Relation: Parts				
P#	P_Name	Color	Weight	City
p_1	Nut	Red	12	London
p_2	Bolt	Green	17	Paris
p_3	Screw	Blue	17	Rome
p_4	Screw	Red	14	London
p_5	Cam	Blue	12	Paris
p_6	Cog	Red	19	London

Relation: Supplied_by		
`S_Num`	P#	Qty
s_1	p_1	300
s_1	p_2	200
s_1	p_3	400
s_1	p_4	200
s_1	p_5	100
s_1	p_6	100
s_2	p_1	300
s_2	p_2	400
s_3	p_2	200
s_4	p_2	200
s_4	p_4	300
s_4	p_5	400

  - *Attributes*
    - Headers in the table
      - `S_Num`, `S_Name`, `Status` and `City` in the relation `Supplier`
  - *Domain*
    - Pool of values from which one or more attributes actually draw their values
    - Smallest semantic unit of data

    * Domain of attribute S_Name above is the set of all suppliers

$$\begin{aligned} D_1 &= \{s_1, s_2, s_3, s_4, s_5\} \\ D_2 &= \{\texttt{Adams, Blake, Clark, Jones, Smith}\} \\ D_3 &= \{10, 20, 30\} \\ D_4 &= \{\texttt{Athens, London, Paris}\} \end{aligned}$$

    * *Atomic domain*
        · Elements of domain are indivisible or nondecomposable
        · No internal structure in the elements
    * *null values* not allowed for domains
    * A domain may contain values that do not appear in any of the tuples at a given instance
  – *Tuple*
    * A row in the table (record)
    * Any row of Supplier must consist of a 4-tuple (a_1, a_2, a_3, a_4) where $a_1 \in D_1$ (S_Num), $a_2 \in D_2$ (S_Name), $a_3 \in D_3$ (Status), and $a_4 \in D_4$ (City)
    * In the table Supplier, there are 5 tuples
    * Denoted by t
        · ith tuple in the table is denoted by t_i
        · $t_5 = (s_5$, Adams, 30, Athens)
        · t_3[City] = Paris
  – Relation
    * Subset of the set of all possible rows
    * The relation Supplier will always be a subset of $D_1 \times D_2 \times D_3 \times D_4$
    * In general, a relation of n attributes must be a subset of

$$r \subset (D_1 \times D_2 \times \cdots \times D_n)$$

    * Mathematically, a relation is a subset of a Cartesian product of a list of domains
    * *Degree*
        · Number of attributes in the relation
    * *Cardinality*
        · Number of tuples in the relation
  – *Primary key*
    * Unique identifier for each tuple in the relation
    * a [combination of] column[s] such that no two tuples in the relation contain the same value in that [combination of] column[s]
  – Notation
    * r – Relation
    * $t \in r$

Database Schema

- A *relation* consists of two parts:
  1. *Database Schema*
     - Logical design of the database
     - Easy to imagine it like a class definition, with tuples being the objects in the class
     - Heading with a fixed set of attributes (or attribute-domain pairs)
     - `Supplier-scheme` = (`S_Num`, *S_Name, Status, City)*
     - Relation `Supplier` is denoted by `Supplier(Supplier-scheme)`
     - Possible to have multiple relations defined on the same schema, such as `employee` and `dependent`
  2. *Database Instance*
     - *Body* of the relation
     - Easy to imagine it like a definition of a set of objects
     - Data in the database at a given time
     - *Time-varying* set of tuples (or attribute-value pairs)
- Keys
  - Same as the E-R model
- Query Languages
  - Language to request information from the database
  - *Procedural Query Language*
    * User instructs the system to perform a sequence of operations on the database to computer the desired result
    * Relational algebra
  - *Nonprocedural Query Language*
    * User just described the information desired without giving a specific procedure to obtain the information
    * Tuple relational calculus
    * Domain relational calculus
  - Data manipulation language

The Relational Algebra

- Collection of high-level operators that operate on relations
- Fundamental operations: select, project, cartesian product, rename, union, set difference
- Other operations: set intersection, natural join, division, assignment
- Property of *closure*

- Output from each operation is another relation
- Output from one relation can become input for another
- Allows for *nested relational expressions*

Fundamental Operations

- The *Select* Operation
  - Select tuples that satisfy a given predicate
  - Denoted by Greek letter σ
  - Predicate appears as a subscript to σ

$$\sigma_{\text{City = ``London''}}(Supplier)$$

  - Result of the above selection

S#	S_Name	Status	City
s_1	Smith	20	London
s_4	Clark	20	London

  - Comparisons are allowed using the operators

$$=, \neq, <, \leq, >, \geq$$

  - Predicates may be combined into larger predicates using the connectives *and* ($\wedge$) and *or* ($\vee$)

- The *Project* Operation
  - Unary operation that returns its argument relation, with certain columns left out
  - Any duplicate rows are eliminated
  - Denoted by Greek letter Π
  - The fields to appear in the final relation are specified as a subscript to Pi

$$\Pi_{\text{S\#, S\_Name}}(Supplier)$$

  - Result of the above projection

S#	S_Name
s_1	Smith
s_2	Jones
s_3	Blake
s_4	Clark
s_5	Adams

  - Combining projection and selection

$$\Pi_{\text{S\_Name, City}}(\sigma_{\text{Status}\leq 20}(Supplier))$$

S_Name	City
Smith	London
Jones	Paris
Clark	London

- The *Cartesian Product* Operation
  - Allows us to combine information from several relations
  - Written as a binary infix operator
  - Denoted by a cross $\times$
  - Attributes common to both relations identified by prefixing them with the relation name followed by a period
  - Query

$$\sigma_{\text{Color = "Red"}}(Supplier \times \text{Parts})$$

S_Num	S_Name	Status	*Supplier.City*	P#	P_Name	Color	Weight	Parts.City
s_1	Smith	20	London	p_1	Nut	Red	12	London
s_2	Jones	10	Paris	p_1	Nut	Red	12	London
s_3	Blake	30	Paris	p_1	Nut	Red	12	London
s_4	Clark	20	London	p_1	Nut	Red	12	London
s_5	Adams	30	Athens	p_1	Nut	Red	12	London
s_1	Smith	20	London	p_4	Screw	Red	14	London
s_2	Jones	10	Paris	p_4	Screw	Red	14	London
s_3	Blake	30	Paris	p_4	Screw	Red	14	London
s_4	Clark	20	London	p_4	Screw	Red	14	London
s_5	Adams	30	Athens	p_4	Screw	Red	14	London
s_1	Smith	20	London	p_6	Cog	Red	19	London
s_2	Jones	10	Paris	p_6	Cog	Red	19	London
s_3	Blake	30	Paris	p_6	Cog	Red	19	London
s_4	Clark	20	London	p_6	Cog	Red	19	London
s_5	Adams	30	Athens	p_6	Cog	Red	19	London

  - If there are n_1 tuples in r and n_2 tuples in s, then the number of tuples in $r \times s$ is $n_1 \cdot n_2$
  - The schema of the cross product is the result of concatenation of the schema of the two relations
- The *Rename* Operation
  - Rename a relation for the query
  - Denoted by ρ
  - $\rho_x(r)$ returns the relation r under the name x
  - $\rho_{x(a_1,a_2,\ldots,a_n)}(E)$ returns the result of the expression E (a relation) with the attributes renamed to $a_1, a_2, \ldots, a_n$
- The *Union* Operation

- Denoted by $\cup$
- Query: "Find all cities."

$$\Pi_{\text{City}}(Supplier) \cup \Pi_{\text{City}}(\text{Parts})$$

City
London
Paris
Athens
Rome

- The relations r and s for $r \cup s$ must be of the same arity
- The domains of ith attribute of r and ith attribute of s must be the same
  * The above two conditions pretty much imply that the schema of relations r and s must be the same

- The *Set Difference* Operator
  - Finds tuples present in one relation but not in the other
  - Denoted by $-$
  - Both relations must be *union-compatible*
  - Query: Find the cities in relation `Parts` that are not in relation `Supplier`

$$\Pi_{\text{City}}(Parts) - \Pi_{\text{City}}(\text{Supplier})$$

City
Rome

Formal Definition of the Relational Algebra

- A basic expression in the relational algebra consists of either of the following:
  - A relation in the database – Time varying tuples
  - A constant relation
- A general expression is constructed out of smaller subexpressions using the relational operators (and predicates, if needed).

Additional Operations

- Provided to simplify common queries
- Can be expressed in terms of fundamental operations
- The *Set Intersection* Operation
  - Denoted by $\cap$

- Defined as

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

- The *Natural Join* Operation
  - Most important of the *join* operations
  - Simplifies queries requiring a Cartesian product
  - Combines certain *selection*s and a *Cartesian product*
  - Denoted by the "join symbol" $\bowtie$
  - Consider relations A and B with schemes

$$A - \text{scheme} = (X_1, X_2, \ldots, X_m, Y_1, Y_2, \ldots, Y_n)$$

$$B - \text{scheme} = (Y_1, Y_2, \ldots, Y_n, Z_1, Z_2, \ldots, Z_p)$$

  - $A \bowtie B$ consists of the following attributes

$$(X_1, X_2, \ldots, X_m, Y_1, Y_2, \ldots, Y_n, Z_1, Z_2, \ldots, Z_p)$$

    Each tuple in $A \bowtie B$ is given by $(X : x, Y : y, Z : z)$ such that a tuple a appears in A with X-value x and Y-value y and a tuple B appears in B with Y-value y and Z-value z
  - Formal definition

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \cdots \wedge r.A_n = s.A_n}(r \times s))$$

    where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.
  - The relation *Supplier* $\bowtie$ *Parts* is given by

S#	S_Name	Status	City	P#	P_Name	Color	Weight
s_1	Smith	20	London	p_1	Nut	Red	12
s_1	Smith	20	London	p_4	Screw	Red	14
s_1	Smith	20	London	p_6	Cog	Red	19
s_2	Jones	10	Paris	p_2	Bolt	Green	17
s_2	Jones	10	Paris	p_5	Cam	Blue	12
s_3	Blake	30	Paris	p_2	Bolt	Green	17
s_3	Blake	30	Paris	p_5	Cam	Blue	12
s_4	Clark	20	London	p_1	Nut	Red	12
s_4	Clark	20	London	p_4	Screw	Red	14
s_4	Clark	20	London	p_6	Cog	Red	19

  - Natural join is associative
    * $(r \bowtie s) \bowtie t \equiv r \bowtie (s \bowtie t) \equiv r \bowtie s \bowtie t$
  - Natural join is commutative
    * $r \bowtie s \equiv s \bowtie r$

- Let $r(R)$ and $s(S)$ be relations without any common attributes, that is $R \cap S = \emptyset$. Then

$$r \bowtie s \equiv r \times s$$

- The *Division* Operator
  - Denoted by $\div$
  - Consider relations A and B with schemes

$$A - \text{scheme} = (X_1, X_2, \ldots, X_m, Y_1, Y_2, \ldots, Y_n)$$

$$B - \text{scheme} = (Y_1, Y_2, \ldots, Y_n)$$

  - Relation A is the *dividend*
  - Relation B is the *divisor*
  - Let $(X_1, X_2, \ldots, X_m)$ and $(Y_1, Y_2, \ldots, Y_n)$ be represented as two composite attributes X and Y
  - The relation $A \div B$ has the scheme X and its instance is the set of all tuples $(X : x)$ such that a tuple $(X : x, Y : y)$ appears in A for all tuples $(Y : y)$ appearing in B
  - Let the dividend be $D = \Pi_{\mathsf{S\#,\ P\#}}\ (Supplied\_by)$
  - Let the divisor be $d = \Pi_{\mathsf{P\#}}(\sigma_{\mathsf{P\#}=p_2 \vee \mathsf{P\#}=p_4}\ (Parts))$
  - $D \div d$ is given by

S#
s_1
s_4

  - In terms of fundamental operations, for relations $r(R)$ and $s(S)$,

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

- The *Assignment* Operation
  - Denoted by $\rightarrow$
  - Allows for temporary assignment of a relation to a variable
  - The query can be written as a sequential program

The Tuple Relational Calculus

- Nonprocedural query language
- Just describe the desired information without giving a specific procedure to obtain that information
- Relational algebra and tuple relational calculus are alternative candidates as the formal basis for the manipulative part of relational model

- Relational algebra provides a collection of explicit operations – join, union, projection, and so on, to tell the system how to build a desired relation from given relations
- Tuple relational calculus merely provides a notation for formulating the definition of the desired relation in terms of available relations
- A query is expressed as $\{t | P(t)\}$
- Value of tuple t on attribute A is denoted by $t[A]$
- Query "Get supplier numbers and cities for suppliers who supply part p_2"
  - Relational algebra
    1. Form the natural join of relations *Supplier* and *Supplied_by*
    2. Select the tuples for part p_2
    3. Project the result of the selection on S# and City
  - Tuple relational calculus

    Get S# and City for suppliers such that there exists a shipment SP with the same S# value and with P# value p_2

    $$\{t | \exists \text{ SP } \in \textit{ Supplied\_by } (t[S\#] = \text{SP}[S\#] \wedge \text{SP}[P\#] = p_2 \wedge t[City] = \textit{ Supplier}[City])\}$$
- Relational algebra and tuple relational calculus are precisely equivalent to each other
  - For every expression of the algebra, there is an equivalent expression in the calculus
- *Tuple variable* or *range variable*
  - Tuple variable *ranges over* some relation
  - Used to represent any tuple of a relation
- More Example queries
  - "Get supplier numbers for suppliers in London"
  - $$\{t | \exists s \in \textit{ Supplier } (t[S\#] = s[S\#] \wedge s[City] = \text{“}London''))\}$$
- Formal Definition
  - *Expression*
    * $\{t | P(t)\}$
    * P is a *formula*
    * t is a *tuple variable*
    * A tuple variable is said to be a *free variable* unless quantified by $\exists$ or $\forall$
    * A quantified variable is said to be *bound*

- A formula is built up out of *atoms* with one of the following forms
  - $s \in r$
    - s is a tuple variable and r is a relation
    - $\notin$ operation is not allowed
  - $s[x]\Theta u[y]$
    - s and u are tuple variables
    - x and y are attributes from the schemes of s and u, respectively
    - Θ is a comparison operator from the set $\{<, \leq, =, \neq, >, \geq\}$
    - SP[S#] $=$ *Supplier*[S#] $\mathcal{F}$[1]
    - SP[P#] $\neq$ *Parts*[P#] $\mathcal{F}$
  - $s[x]\Theta c$
    - s is a tuple variable
    - x is an attribute from the scheme of s
    - Θ is a comparison operator from the set $\{<, \leq, =, \neq, >, \geq\}$
    - c is a constant from the domain of attribute x
    - SP[S#] $=$ "s_1" $\mathcal{F}$
- Formulae are built up from atoms using the following rules
  - An atom is a formula
  - If P_1 is a formula, then so are $\neg P_1$ and (P_1)
    - $\neg$ (SP[City] $=$ "London")
  - If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$
    - SP[S#] $=$ *Supplier*[S#] $\wedge$ SP[P#] $\neq$ *Parts*[P#]
  - If $P_1(s)$ is a formula containing a free tuple variable s, then

    $$\exists s \in r(P_1(s))$$

    and

    $$\forall s \in r(P_1(s))$$

    are also formulae
    - $\exists$SP (SP[S#] $=$ *Supplier*[S#] $\wedge$ SP[P#] $=$ "p_2")
- Equivalence in expressions

$$\begin{array}{lcl} P_1 \wedge P_2 & \Leftrightarrow & \neg(\neg P_1 \vee \neg P_2) \\ \forall t \in r(P_1(t)) & \Leftrightarrow & \neg \exists t \in r(\neg P_1(t)) \\ P_1 \Rightarrow P_2 & \Leftrightarrow & \neg P_1 \vee P_2 \end{array}$$

- Safety of Expressions
  - Possible to generate infinite relations
    - $\{t | \neg t \in \; Supplier\}$

[1] $\mathcal{F}$ = free occurence of all tuple variables

- *Domain* of a tuple relational formula
- An expression $\{t|P(t)\}$ is *safe* if all values in the result are values from dom(P)

- Expressive power of languages – converting relational algebra expressions into equivalent tuple calculus expressions.
  - Selection σ
    - Relational algebra –
$$\sigma_{A\theta c}(R)$$
$$\sigma_{A\theta B}(R)$$
    - Tuple calculus –
$$\{x(R)|f(x) \wedge x(A)\theta c\}$$
$$\{x(R)|f(x) \wedge x(A)\theta x(B)\}$$
  - Projection Π
    - Relational algebra –
$$\Pi_{a_1a_2\ldots a_n}(R)$$
    - Tuple calculus –
$$\{x(a_1a_2\ldots a_n)|\exists y \in R \wedge x(a_i) = y(a_i)\}$$
  - Join $\bowtie$
    - Relational algebra –
$$R(QRS) = R_1(QR) \bowtie R_2(RS)$$
    - Tuple calculus –
$$\{z(QRS)|\exists x(QR)\exists y(RS)(f(x) \wedge g(y) \wedge z(QR) = x(QR) \wedge z(RS) = y(RS))\}$$
  - Other operations left as an exercise
  - Examples
    - Example 1
      - Relation algebra
$$\Pi_{\textsf{sub\_part\_of nused}}(\mathit{pinfo} \ \bowtie \sigma_{\textsf{ptype = 747}}(\mathit{used\_on}))$$
      - Tuple calculus
$$\{w(\textsf{subpart nused})|\exists z(\textsf{part\# sub\_part\_of partname ptype nused})(\exists x(R_1)\exists y(R_2))$$

<u>The Domain Relational Calculus</u>

- Closely related to tuple relational calculus
- Uses *domain variables* to represent single domain values instead of tuple variables (that represent entire tuple)

- An expression in domain calculus is of the form

$$\{\langle x_1, x_2, \ldots, x_n\rangle \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula composed of atoms
  - An atom has one of the following forms:
    * $\langle x_1, x_2, \ldots, x_n\rangle \in r$, where r is a relation on n attributes and $x_1, x_2, \ldots, x_n$ are domain variables or domain constants
    * $x \Theta y$, where x and y are domain variables and Θ is a comparison operator
    * $x \Theta c$
  - Formulae are built up from atoms using the following rules:
    * An atom is a formula
    * If P is a formula then so are $\neg P$ and (P)
    * If P_1 and P_2 are formulas then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$
    * If $P(x)$ is a formula in x, where x is a domain variable, then $\exists x(P(x))$ and $\forall x(P(x))$ are also formulas

- Example queries

  - "Get supplier numbers for suppliers in London"
  -

$$\{\langle x\rangle \mid \exists y \exists z(Supplier(x\ y\ z\ \text{"London"}))\}$$

- Safety of expressions

  - Consider the expression

$$\{\langle x\rangle \mid \exists y(\langle x\ y\rangle \in r) \wedge \exists z(\neg(\langle x\ z\rangle \in r))\}$$

    We must consider values that do not appear in r
    Not allowed
  - An expression

$$\{\langle x_1, x_2, \ldots, x_n\rangle \mid P(x_1, x_2, \ldots, x_n)\}$$

    is safe if all of the following hold:
    * All values that appear in tuples of the expression are values from $dom(P)$
    * For every "there exists" subformula of the form $\exists x(P(x))$, the subformula is true if and only if there is a value x in $dom(P)$ such that $P(x)$ is true
    * For every "for all" subformula of the form $\forall x(P(x))$, the subformula is true if and only if there is a value x in $dom(P)$ such that $P(x)$ is true

- Expressive Power of Languages

- If safe, the domain calculus expression is equivalent in expressive power to tuple calculus and relational algebra

Modifying the Database

- Deletion
  - Remove the selected tuples from the database
  - Can only delete whole tuples
  - Expressed as
    $$r \leftarrow r - E$$
    where E is a relational algebra query
  - "Delete all of Smith's accounts."
    $$deposit \leftarrow deposit - \sigma_{\text{customer-name = "Smith"}}(deposit)$$
- Insertion
  - Insert a new tuple into the database
  - Attribute values must be members of the attribute's domain
  - Expressed as
    $$r \leftarrow r \cup E$$
    where E is a relational algebra expression
  - Insert the fact the a new customer Smith has $1200 in account 9732 at the Perryridge branch
    $$deposit \leftarrow deposit \cup \{(\text{"Perryridge"}, 9732, \text{"Smith"}, 1200)\}$$
- Updating
  - Change soem values in a tuple without changing *all* the values
  - **update** operator δ
  - Expressed as
    $$\delta_{A \leftarrow E}(r)$$
    where attribute A is assigned the value of expression E in relation r
  - Let accounts with balances over $10,000 receive 6% interest and others receive 5% interest. The new balance can be expressed as
    $$\delta_{\text{balance} \leftarrow \text{balance} \times 1.06}(\sigma_{\text{balance} > 10000}(deposit))$$
    $$\delta_{\text{balance} \leftarrow \text{balance} \times 1.05}(\sigma_{\text{balance} \leq 10000}(deposit))$$

Views

- Virtual relation constructed by using projection and other operators
- Definition of view must be stored as part of the database
- View definition
  - **create view** statement

    $$\text{create view } v \text{ as } \langle\text{query expression}\rangle$$

  - View made up of all branches and their customers

    $$\begin{array}{ll}\text{create} & \text{view } \textit{all-customer as} \\ & \Pi_{\text{branch-name, customer-name}}(\textit{deposit}) \\ & \cup\, \Pi_{\text{branch-name, customer-name}}(\textit{borrow})\end{array}$$

- Updates through views
  - Views represent only a part of the tuple in the original relation
  - Insertion of new tuple may lead to null values in the original relation
  - Solution – put a place-holder *null* in the database