Relational Model: History

• Objectives of Relational Model:
  1. Promote high degree of data independence
  2. Eliminate redundancy, consistency, etc. problems
  3. Enable proliferation of non-procedural DML’s

• Relational Model History
  1. System R
     – Late 70’s, IBM
     – Descendents: SQL, DB2, Oracle
  2. Interactive Graphics Retrieval System (INGRES)
     – Berkeley
     – Descendents: INGRES
  3. Peterlee Relational Test Vehicle
     – IBB (UK)
     – Components
       * Structural part
       * Integrity rules
       * DML

Relational Model: Formal Query Language - Relational Algebra

• Procedural language
• Returns sets of records
• Consists of 5 basic operations
• Arguments to operators are relations, and operators produce relations as results
  – I.e., relations are closed under the relational algebra
  – Allows nesting of operators

• Operators:
  1. Select
     – Operator: \( \sigma \)
     – Cardinality: unary
     – Syntax: \( \sigma_p(relational\_expression) \)
     – Semantics: Returns those tuples in argument \( relational\_expression \) that meet the conditions specified by predicate \( p \)
       * \( p \) specified in terms of Boolean operators, column names, and values
2. Project
   - Operator: $\pi$
   - Cardinality: unary
   - Syntax: $\pi_{\text{attribute list}}(\text{relational expression})$
   - Semantics: Returns a subtuple for each tuple in argument $r$, using only the columns specified in $\text{attribute list}$

3. Cartesian product
   - Operator: $\times$
   - Cardinality: binary
   - Syntax: $r \times s$
   - Semantics: Resulting tuples generated by pairing every tuple in argument $r$ with every tuple in argument $s$
     \[ r \times s \equiv \{ <a_1, ..., a_m, b_1, ..., b_n> | <a_1, ..., a_m> \in r \land <b_1, ..., b_n> \in s \} \]
   - If $R$ has $m$ columns and $S$ has $n$ columns, $r(R) \times s(S)$ has $m + n$ columns
   - If $r$ has $i$ rows and $s$ has $j$ rows, $r \times s$ has $i \times j$ rows
   - Columns labeled using $\text{Schema name.column name}$
   - Rarely used in isolation
     * Usually only the resulting rows that represent info re the same entity are retained, via a selection operation
     * Operation in which two or more related tables are combined and only the rows pertaining to a single entity are retained is called a join
4. Union
   - Operator: $\bigcup$
   - Cardinality: binary
   - Syntax: $r \bigcup s$
   - Semantics: Standard set union operation of arguments $r(R)$ and $s(S)$
   - In order to perform union, arguments must be union compatible:
     (a) $arity(R) = arity(S)$
     (b) If $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$ then,
        $domain(A_i) \equiv domain(B_i), 1 \leq i \leq n$

5. Set difference
   - Operator: $-$
   - Cardinality: binary
   - Syntax: $r - s$
   - Semantics: Standard set difference operation of arguments $r$ and $s$
   - $r$ and $s$ must be union compatible

Relational Model: Formal Query Language - Relational Algebra Extensions

- The original language has been extended to facilitate queries that would otherwise be complex
- Extended relational algebra operations:

1. Assignment
   - Operator: $\leftarrow$
   - Cardinality: binary
   - Syntax: $variable \leftarrow relational\_expression$
   - Semantics: Standard variable assignment
     * Allows temporary storage of intermediate results
2. Renaming columns
   - Syntax: `variable(A_1, A_2, ..., A_n) ⤷ relational_expression`
   - Arity of `relational_expression` must be `n`
   - Semantics: Assigning alias to a column name
     * Simplifies some queries, especially recursive ones

3. Renaming relations (on-the-fly)
   - Syntax: `ρ_{rel \_name}(relational \_expression)`

4. Theta join
   - Operator: `⋈_θ`
   - Cardinality: binary
   - Syntax: `r ⋈_θ s`
   - Semantics: Equivalent to a cartesian product of its arguments followed by a selection based on the specified predicate
     * Given `r(R)` and `s(S)`,
       `r ⋈_θ s \equiv σ_θ(r \times s))`
     * Result has a column for each schema column of argument `r` and argument `s`

5. Equijoin
   - Same as theta join except predicate restricted to equality test on columns
6. Natural join

- Operator: *
- Cardinality: binary
- Syntax: \( r \ast s \)
- Semantics: Equivalent to an equijoin except that no predicate is specified
  * Given \( r(R), s(S), T = R \cup S, I = R \cap S \)
  \[ r \ast s \equiv \pi_T(\sigma_{r.a_1=s.a_1, r.a_2=s.a_2, \ldots, r.a_n=s.a_n}(r \times s)), \text{ where } a_i \in I \]
  * Equivalency based on columns in schemas of arguments \( r \) and \( s \) that have the same name
  * Only rows that have the same values for these same-named columns will be retained
  * The result will only have one copy of the same-named columns

7. Semijoin

- Operator: \( \bowtie_p \) or \( \bowtie_p \)
- Cardinality: binary
- Syntax: \( r \bowtie_p s \)
- Semantics: Returns those tuples in \( r \) that would participate in a join with \( s \)
  * Given \( r(R), s(S), \)
  \[ r \bowtie_p s \equiv \pi_R(\sigma_p(r \times s)) \]
8. Division
   - Operator: ÷
   - Cardinality: binary
   - Syntax: \( r \div s \)
   - Semantics:
     * Given \( r(R), s(S) \), \( S \subset R, T = R - S \) (i.e., \( T \) are those columns that appear in \( R \) but not in \( S \))
     * Then
       (a) \( r \div s \equiv t \) will consist of columns \( T \)
       (b) There will be one tuple in the result for each tuple in \( r \) that is paired with every set of values for \( R \) that appear in relation \( s \)
     * \( r \div s \equiv t \) where \( t \) computed as
       (a) \( t_1 = \pi_T(r) \)
       (b) \( t_2 = \pi_T((t_1 \times s) - r) \)
       (c) \( t = t_1 - t_2 \)

   ![Diagram](image)

   Relational Model: Formal Query Language - Relational Algebra Enhancements

   - The original language has also been extended to facilitate queries that would otherwise be impossible
   - Relational algebra enhancements:
     1. Aggregate functions
        - Operator: \( gF_f \), where
          * \( g \) is a list of attribute names
          * \( f \) is a list of function-attribute name pairs
        - Cardinality: unary
        - Syntax: \( gF_f(r) \)
        - Semantics: Function \( f \) is applied to columns \( c \) in argument \( r \)
          * Functions in \( f \) are range functions
            - They perform a calculation on a range of values (Max, Avg, ...)
            - Each function in \( f \) is applied to all tuples in \( r \)
            - Each function is applied only to the attribute it is paired with
            - The resulting relation will have one column for each function
          * May include optional grouping attributes \( g \)
            - If \( g \) appears, then tuples in \( r \) are grouped in terms of the unique attribute values for each attribute occurring in \( g \), on an attribute-by-attribute basis
            - The functions in \( f \) are then applied to each group of tuples
            - The result will contain 1 tuple for each group
2. Outer joins

- Operator: $\Pi_p$ (left outer join), $\Pi_E_p$ (right outer join), $\Pi_E_p$ (full outer join)
- Cardinality: binary
- Syntax: $r \Pi_p s$, etc.
- Semantics: Same as a theta join, except:
  * Includes those tuples in argument $r$ that have no match in argument $s$, with unmatched fields of $r$ padded with NULL
  * (Right and full outer joins similar)

3. Outer union

- Operator: $\cup$ (Note: No formal symbol for this)
- Cardinality: binary
- Syntax: $r \cup s$
- Semantics:
  * Let $r(R)$, $s(S)$, $U = R \cup S$, and $R \cap S \neq \emptyset$
  * $R \cap S$ must contain keys of $R$ and $S$
  * Then $u(U) = r \cup s$, generated as
    - If $t_i \in r$ has same key value as $u_j \in s$, generate new tuple in $u$ with attributes of $t_i$ and $u_j$
    - For tuples in $r$ and $s$ that do not meet above requirement, include a tuple for each with attributes from other relation padded with NULL
  * Allows "union" of non-union compatible relations
4. Recursive closure
   – Applicable to recursive relations
   – Apply a relation recursively to the results of its application
   – Only achieved using recursion in a host programming language

Relational Model: Codd’s Rules of Relational DBMSs

• Foundational Rules

Rule 0: Foundational rule For any system that is advertised as, or claimed to be, a relational DBMS, that system must be able to manage DB’s entirely through its relational capabilities.

Rule 12: Nonsubversion rule If a relational system has a low-level (record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level (set-at-a-time) relational language.

• Structural Rules

Rule 1: Information representation All information in a relational DB is represented explicitly at the logical level and in exactly one way – by values in tables.

Rule 6: View update All views that are theoretically updatable are also updatable by the system.

• Integrity rules

Rule 3: Systematic treatment of NULLs NULL values are supported for representing missing information and inapplicable information in a systematic way, independent of data type.

Rule 10: Integrity independence Integrity constraints specific to a particular relational DB must be definable in the relational DB sublanguage and storable in the catalog, not in application programs.

• Data Manipulation Rules

Rule 2: Guaranteed access Each and every datum in a relational DB is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.

Rule 4: Dynamic online catalog based on relational model The DB description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

Rule 5: Comprehensive data language A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements can express all of the following items.
   – data definition
   – view definition
   – data manipulation
   – integrity constraints
   – authorization
   – transaction boundaries

Rule 7: High-level insert, update, delete The capability of handling a base relation or derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data.

• Data Independence Rules

Rule 8: Physical data independence Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

Rule 9: Logical data independence Application programs and terminal activities remain logically unimpaired whenever information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.
**Rule 11: Distribution independence** The data manipulation sublanguage of a relational DBMS must enable application programs and inquiries to remain logically the same whether data are physically centralized or distributed.

---

**Relational Model: ER to RM Mapping**

1. Mapping "regular" (strong) ETs
   - Create a table for each regular ET
   - Columns represent attributes
   - If attribute composite, include only leaves
   - Do not include multivalued attributes (see later)

2. Mapping weak ETs
   - Create a table for each weak ET
   - Proceed as for strong ETs
   - Include columns for key attributes of parent ET (becomes foreign key)
   - Table key is union of parent key and discriminator

3. Mapping 1:1 binary relations
   - **Foreign key approach**
     - Preferred method
     - Identify ET that has total participation
       - Include columns for key attributes of the other ET (becomes foreign key)
       - Include columns for any attributes of the relation type itself
     - If neither ET has total participation
       - Do as above
       - Note that result will include NULL (foreign key attributes) for some tuples
     - Could include key attributes of each ET in the other
       - Compounds redundancy problem
   - **Merged relation approach**
     - Most relevant if both ETs participate totally
     - Combine 2 ETs into single table
     - Include columns for any attributes of the relation type itself
     - Table key is key of either participant
   - **Cross reference (relationship relation) approach**
     - Create table consisting of
       i. Key attributes of each participant (become foreign keys)
       ii. Attributes of the relation type itself
     - Table key is key of either participant, providing no NULL values
     - Also referred to as a lookup table

4. Mapping 1:N binary relations
   - Include columns for key attributes of the unary ET in the table of the n-ary participant (becomes foreign key)
   - Include columns for any attributes of the relation type itself
   - Alternative is cross reference approach as described in 1:1 case
5. Mapping N:M binary relations
   - Create a table consisting of
     (a) Key attributes of each participant (become foreign keys)
     (b) Attributes of the relation type itself
   - Table key is union of participant keys

6. Mapping multivalued attributes
   - Create a table consisting of
     (a) Key attributes of the parent ET (becomes foreign key)
     (b) The multivalued attribute
   - Table key is entire set of attributes

7. Mapping n-ary relations
   - Handle as for binary N:M relation types
   - Table key is generally union of keys of participants
   - The key of any unary participant does not become part of key of new relation

8. Mapping specialization/generalization
   (a) Multiple relations: super- and sub- classes approach
      - Create tables for the superclass and for each subclass
      - Include in each subclass relation the key of the superclass (becomes foreign key)
      - Table key of subclass is key of superclass
      - Most general approach - works for all types of constraints
   (b) Multiple relations: subclasses only approach
      - Create tables for each subclass
      - Attributes of each child are the union of its attributes and the parent’s attributes
      - Table key of subclass is key of superclass
      - Applicable only if superclass participates totally
        - If not, superclass entities that are not members of subclass will be lost
      - For overlapping subclasses, will duplicate info
      - No relation holds all of the superclass entities
        - Would need to perform outer union or full outer join to recover
(c) Single relation: 1 type attribute approach

- Create a table consisting of
  i. Attributes of each subclass
  ii. Attributes of the superclass
  iii. A new type attribute (discriminator)
      - Value of discriminator indicates which subclass an entity belongs to
      - If specialization is attribute-defined, do not include
        Defining attribute value can be used for discrimination
- Attributes of each child are the union of its attributes and the parent’s attributes
- Table key is key of superclass
- Applicable only if disjoint specialization
- Results in NULL values for subclass-specific attributes to which entity does not belong
- Preferred approach if few attributes associated with each subclass

(d) Single relation: multiple type attributes approach

- Create a table as for single relation: 1 type attribute above
- Include one type attribute for each subclass
  - Discriminators are Boolean
- Most applicable for overlapping specialization (but works for disjoint)

9. Mapping unions

- Create a table for the union type
- Key handled in one of following ways:
  (a) If all categories have same key
      - Include key in union table
  (b) If all categories have different keys
      - Include a new surrogate key attribute in union type
      - Is artificial attribute whose value is guaranteed to be unique
      - Include surrogate key in each category as well
      - Key will be NULL for category entities that do not participate in union