

Databases Illuminated

Chapter 4

The Relational Model

History of Relational Model

- 1970 Paper by E.F. Codd “A Relational Model of Data for Large Shared Data Banks” proposed relational model
- System R, prototype developed at IBM Research Lab at San Jose, California – late 1970s
- Peterlee Test Vehicle, IBM UK Scientific Lab
- INGRES, University of California at Berkeley
- System R results used in developing DB2 from IBM and also Oracle
- Early microcomputer based DBMSs were relational - dBase, R;base, Paradox
- Microsoft’s Access, now most popular microcomputer-based DBMS, is relational
- Oracle, DB2, Informix, Sybase, Microsoft’s SQL Server, MySQL - most popular enterprise DBMSs, all relational

Advantages of Relational Model

- Based on mathematical notion of relation
- Can use power of mathematical abstraction
- Can develop body of results using theorem and proof method of mathematics – results then apply to many different applications
- Can use expressive, exact mathematical notation
- Theory provides tools for improving design
- Basic structure is simple, easy to understand
- Separates logical from physical level
- Data operations easy to express, using a few powerful commands
- Operations do not require user to know storage structures used

Data Structures

- Relations are represented physically as tables
- Tables are related to one another
- Table holds information about objects
- Rows (tuples) correspond to individual records
- Columns correspond to attributes
- A column contains values from one domain
- Domains consist of atomic values

Properties of Tables

- Each cell contains at most one value
- Each column has a distinct name, the name of the attribute it represents
- Values in a column all come from the same domain
- Each tuple is distinct – no duplicate tuples
- Order of tuples is immaterial

Example of Relational Model

- See Figure 4.1
- Student table tells facts about students
- Faculty table shows facts about faculty
- Class table shows facts about classes, including what faculty member teaches each
- Enroll table relates students to classes

Mathematical Relations

- For two sets D_1 and D_2 , the **Cartesian product**, $D_1 \times D_2$, is the set of all ordered pairs in which the first element is from D_1 and the second is from D_2
- A **relation** is any subset of the Cartesian product
- Could form Cartesian product of 3 sets; relation is any subset of the ordered triples so formed
- Could extend to n sets, using n -tuples

Database Relations

- A **relation schema**, \mathbf{R} , is a set of attributes A_1, A_2, \dots, A_n with their domains D_1, D_2, \dots, D_n
- A **relation** r on relation schema \mathbf{R} is a set of mappings from the attributes to their domains
- r is a set of n -tuples $(A_1:d_1, A_2:d_2, \dots, A_n:d_n)$ such that $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n$
- In a table to represent the relation, list the A_i as column headings, and let the (d_1, d_2, \dots, d_n) become the n -tuples, the rows of the table

Properties of Relations

- **Degree:** the number of attributes
 - 2 attributes - binary; 3 attributes - ternary; n attributes - n-ary
 - A property of the intension – does not change
- **Cardinality:** the number of tuples
 - Changes as tuples are added or deleted
 - A property of the extension – changes often
- **Keys**
- **Integrity constraints**

Relation Keys

- Relations never have duplicate tuples, so you can always tell tuples apart; implies there is always a key (which may be a composite of all attributes, in worst case)
- **Superkey**: set of attributes that uniquely identifies tuples
- **Candidate key**: superkey such that no proper subset of itself is also a superkey (i.e. it has no unnecessary attributes)
- **Primary key**: candidate key chosen for unique identification of tuples
- Cannot verify a key by looking at an instance; need to consider semantic information to ensure uniqueness
- A **foreign key** is an attribute or combination of attributes that is the primary key of some relation (called its **home** relation)

Integrity Constraints

- Integrity: correctness and internal consistency
- **Integrity constraints** are rules or restrictions that apply to all instances of the database
- Enforcing integrity constraints ensures only legal states of the database are created
- Types of constraints
 - **Domain constraint** - limits set of values for attribute
 - **Entity integrity**: no attribute of a primary key can have a null value
 - **Referential integrity**: each foreign key value must match the primary key value of some tuple in its home relation or be completely null.
 - **General constraints** or **business rules**: may be expressed as table constraints or assertions

Representing Relational Database Schemas

- Can have any number of relation schemas
- For each relation schema list name of relation followed by list of attributes in parentheses
- Underline primary key in each relation schema
- Indicate foreign keys (We use italics – arrows are best)
- Database schema actually includes domains, views, character sets, constraints, stored procedures, authorizations, etc.
- Example: University database schema
 - Student (stuld, lastName, firstName, major, credits)
 - Class (classNumber, *facId*, schedule, room)
 - Faculty (facId, name, department, rank)
 - Enroll(*stuld*,*classNumber*,grade)

Types of Relational Data Manipulation Languages

- **Procedural:** proscriptive - user tells system how to manipulate data - e.g. relational algebra
- **Non-procedural:** declarative - user tells what data is needed, not how to get it - e.g. relational calculus, SQL
- Other types:
 - **Graphical:** user provides illustration of data needed e.g. Query By Example(QBE)
 - **Fourth-generation:** 4GL uses user-friendly environment to generate custom applications
 - **Natural language:** 5GL accepts restricted version of English or other natural language

Relational Algebra

- Theoretical language with operators that apply to one or two relations to produce another relation
- Both operands and results are tables
- Can assign name to resulting table (rename)
- SELECT, PROJECT, JOIN allow many data retrieval operations

SELECT Operation

- Applied to a single table, returns rows that meet a specified predicate, copying them to new table
- Returns a horizontal subset of original table

SELECT *tableName* WHERE *condition* [GIVING *newTableName*]

Symbolically, [*newTableName* =] $\sigma_{\text{predicate}}$ (*table-name*)

- Predicate is called **theta-condition**, as in σ_{θ} (*table-name*)
- Result table is horizontal subset of operand
- Predicate can have operators <, <=, >, >=, =, <>, & (AND), | (OR), ~ (NOT)

PROJECT Operator

- Operates on single table
- Returns unique values in a column or combination of columns

PROJECT *tableName* OVER (*colName*,...,*colName*) [GIVING *newTableName*]

Symbolically

[*newTableName* =] ? *colName*,...,*colName* (*tableName*)

- Can compose SELECT and PROJECT, using result of first as argument for second

Product and Theta-join

- Binary operations – apply to two tables
- **Product:** Cartesian product – cross-product of A and B – A TIMES B, written $A \times B$
 - all combinations of rows of A with rows of B
 - Degree of result is deg of A + deg of B
 - Cardinality of result is (card of A) * (card of B)
- **THETA join:** TIMES followed by SELECT
 $A \bowtie_{\theta} B = \sigma_{\theta}(A \times B)$

Equi-join and Natural Join

- **EQUIJOIN** formed when ? is equality on column common to both tables (or comparable columns)
- The common column appears twice in the result
- Eliminating the repeated column in the result gives the **NATURAL JOIN**, usually called just JOIN

A |x| B

Semijoins and Outerjoins

- **Left semijoin** $A \bowtie B$ is formed by finding $A \bowtie B$ and projecting result onto attributes of A
- Result is tuples of A that participate in the join
- **Right semijoin** $A \bowtie B$ defined similarly; tuples of B that participate in join
- **Outerjoin** is formed by adding to the join those tuples that have no match, adding null values for the attributes from the other table e.g.

A OUTER-EQUIJOIN B

consists of the equijoin of A and B, supplemented by the unmatched tuples of A with null values for attributes of B and the unmatched tuples of B with null values for attributes of A

- Can also form **left outerjoin** or **right outerjoin**

Division

- Binary operator where entire structure of one table (divisor) is a portion of structure of the other (dividend)
- Result is projection onto those attributes of the dividend that are not in the divisor of the tuples of dividend that appear with all the rows of the divisor
- It tells which values of those attributes appear with all the values of the divisor

Set Operations

- Tables must be **union compatible** – have same basic structure
- A **UNION** B: set of tuples in either or both of A and B, written $A \cup B$
- A **INTERSECTION** B: set of tuples in both A and B simultaneously, written $A \cap B$
- Difference or A **MINUS** B: set of tuples in A but not in B, written $A - B$

Relational Calculus

- Formal non-procedural language
- Two forms: **tuple-oriented** and **domain-oriented**
- Based on predicate calculus

Tuple-oriented predicate calculus

- Uses tuple variables, which take tuples of relations as values
- Query has form $\{S \mid P(S)\}$
 - S is the tuple variable, stands for tuples of relation
 - $P(S)$ is a formula that describes S
 - Means “Find the set of all tuples, s , such that $P(S)$ is true when $S=s$.”
- Limit queries to **safe expressions** – test only finite number of possibilities

Domain-oriented predicate calculus

- Uses variables that take their values from domains
- Query has form
$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_m) \}$$
 - x_1, x_2, \dots, x_n are domain variables
 - $P(x_1, x_2, \dots, x_m)$ is a predicate
 - $n \leq m$
 - Means set of all domain variables x_1, x_2, \dots, x_n for which predicate $P(x_1, x_2, \dots, x_m)$ is true
- Predicate must be a formula
- Often test for membership condition, $\langle x, y, z \rangle \in X$

Views

- External models in 3-level architecture are called external views
- Relational views are slightly different
- Relational view is constructed from existing (base) tables
- View can be a window into a base table (subset)
- View can contain data from more than one table
- View can contain calculated data
- Views hide portions of database from users
- External model may have views and base tables

Mapping ER to Relational Model

- Each strong entity set becomes a table
- Non-composite, single-valued attributes become attributes of table
- Composite attributes: either make the composite a single attribute or use individual attributes for components, ignoring the composite
- Multi-valued attributes: remove them to a new table along with the primary key of the original table; also keep key in original table
- Weak entity sets become tables by adding primary key of owner entity
- Binary Relationships:
 - 1:M-place primary key of 1 side in table of M side as foreign key
 - 1:1- make sure they are not the same entity. If not, use either key as foreign key in the other table
 - M:M-create a relationship table with primary keys of related entities, along with any relationship attributes
- Ternary or higher degree relationships: construct relationship table of keys, along with any relationship attributes