Chapter 7: The Relational Data Model and Relational Algebra

Relational model developed by Codd (1970)
- popular because
    - formal model
    - simple and unifor structure
    - good compromise between physical db and “real world” domain

Relational Model Concepts:
- relational model represents database as a collection of relations
- informally thought of like a table - rows and columns (some differences we will see)
- domain D is a set of atomic values (indivisible as far as model is concerned)
    - can specify a data type from which the data values are drawn
    - examples: Names, Prices, Dates
    - data type or format also specified for each domain
        - Names: string of characters up to 20 long
        - Prices: real number between 0.00 and 1000.00
        - Dates: char string of the form dd/dd/dd where each d is a numeric digit and together they form a valid calendar date

- relation schema R $\qquad$ $R(A_1, A_2, \ldots, A_n)$
    - made up of relation name R and list of attributes $A_1, A_2, \ldots, A_n$
    - each attribute A_i is the name of a role played by some domain D in R
        - D is called the domain of A_i - $dom(A_i)$
    - example:

        TOY(TOY_NUM, NAME, MAN_ID, MSRP, AGE_GROUP, NUM_IN_STOCK, SOLD_YTD)
        - TOY is the name of the relation
        - 7 attributes
        - domains: dom(TOY_NUM) = number between 000 and 999

          dom(NAME) = Names

          dom(MSRP) = Prices, etc.

- relation (relation instance) r of R (denoted r(R))
    - a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$
        - each n-tuple is an ordered list of n values $t = <v_1, v_2,\ldots, v_n>$
            - v_i is an element of $dom(A_i)$ or is null

    - alternate def: r(R) is a subset of the Cartesian product of the domains that define R
        - r(R) subset $(dom(A_1) \times dom(A_2) \times \ldots dom(A_n))$
        - specifies all possible combinations of values from the domains

    - current relation state -reflects the current state of the real world - the current values of the tuples in the relation
        - can change as the state of the environment changes

    - relation schema is relatively static

Notes: (how is a relation different from a file?)

1) Tuples in a relation do not have any particular order
   - can be viewed in order to make logical sense
   - the file in which the relation is stored is physically ordered

2) Logical ordering of attributes in the schema is not important as long as the corresponding values in the relation instance are in the same order
   - define r(R) as a finite set of mappings $r = \{t_1, t_2, ..., t_m\}$
     each ti is a mapping from $R = \{A_1, A_2,..., A_n\}$ to
     $D = dom(A_1)$ union $dom(A_2)$ union ... union $dom(A_n)$
   - each mapping is a tuple
   - tuple is a set of (<attribute>, <value>) pairs
   - ordering of attributes not important
   - using the previous def on a relation in which attributes are ordered is simpler - this mapping definition is more general

3) Each value in a tuple is atomic - not divisible into parts
   - First normal form assumption
   - needed for formal representation of relational databases
   - later research addressed the possibility of composite attributes - now some db's allow it

4) Several interpretations of a relation:
   - AI: relation schema as an assertion and each tuple as a fact about the object or relationship
     - ex: schema of CUSTOMER asserts that each customer has a number, name, address, etc.
       - the first tuple in the file asserts the fact that Karen Smith lives at 42 Kenmore Rd and has 4 children
   - logic: relation is a predicate
     - the values in each tuple are interpreted as values that satisfy the predicate
     - ex: stating TOY(011,Farm House, FP, 29.95,...) implies that this predicate is true

Given all of the possible ways of expressing and interpreting a relation, we explicitly specify our notation that we will use throughout the chapter

- relation schema of degree n: $R(A_1, A_2, ..., A_n)$
- n-tuple t in r(R): $t = <v_1, v_2,...,v_n>$, v_i value wrt A_i
- $t[A_i]$: value in t for A_i
- $t[A_u, A_w, ..., A_z]$: subtuple of values in t correspoinding to the attributes listed
- Q, R, S denote relation names
- q, r, s denote relation states
- t, u, v denote tuples
- name of tuple (ORDER) refers to the current relation state; name with attribute names (ORDER(Order_Num, Cust_Num, Toy_Num,...)) refers to the relation schema
- dot notation may be used: ORDER.Order_Num

Ex: tuple t = <5035, 002, 325, 2, 08/21/95, 12 Eastgate Ames, IA, 00/00/00>; t[Date_Ordered] = <08/21/95>;
t[Date_Ordered, Deliv] = <08/21/95,00/00/00>

Key attributes of a relation:

- No two tuples of a relation may have exactly the same values - must be distinct
- superkey of relation R: a subset of attributes of a relation R such that no two tuples in any relation instance r of R has the same combination of values for the attributes in the subset
  - let SK denote such a subset, then for any tuples $t_1 <> t_2$, $t_1[SK] <> t_2[SK]$
  - ex: from TOY db - SK = {TOY_NUM, SOLD_YTD}
- key of relation R: a superkey of R such that removing any attribute results in a set that is not a superkey (minimal superkey)
  - ex: the above SK is a superkey of the TOY relation, but it is not a key because removing SOLD_YTD would yield another superkey
- the value of a key attribute uniquely identifies a tuple in a relation
- a relation schema can have more than one key
  - each key is called a candidate key
  - primary key is the key used to identify tuples in the relation
    - usually the one with the fewest attributes
    (by convention, primary key is underlined in notation)
  - ex: imagine the CUSTOMER relation has customer number and SS# - both are unique, but for the purposes of this db we use cust num as primary key

Definitions:

- relational database schema S: a set of relation schemas S = {R_1, R_2, ..., R_m} and a set of integrity constraints IC
- relational database DB of S: a set of relation instances DB = {r_1, r_2, ..., r_m} such that each ri is an instance of R_i, and each r_i satisfies constraints in IC

- ex: the Toy Catalog relational database schema includes the relation schemas TOY, CUSTOMER, MANUFACTURER and ORDER
  - the instances of these relations make up the relational database

Constraints in a relational db:

- domain constraints - specify that values of an attribute must be from the specified domain
  ex: MSRP attribute in TOY db must be in the Prices domain
- key constraints - all tuples in a relation must be distinct - existence of at least one key attribute
- entity integrity constraint - no primary key value can be null
- referential integrity constraint - specified between two relations - a tuple in one relation that refers to another relation must refer to an existing tuple i nthat relation
  ex: in the TOY relation, the value of the attribute MAN_ID must exist as a MAN_ID in the MANUFACTURER relation

  - def: a set of attributes FK in relation schema R_1 is a **foreign key** of R_1 if:
    1) attributes of FK have the same domain as the primary key attributes PK of another relation schema R_2
    2) for every tuple t_1 of R_1, either $t_1[FK]$ = null or $t_1[FK]=t_2[PK]$ for some t_2 of R_2

ex: in the above example, MAN_ID is a foreign key of the TOY relation refering to the primary key MAN_ID in the MANUFACTURER relation
- note: the keys do not have to have the same name, but it is allowed

(draw a picture to show the referential integrity constraints in the toy catalog database)

Update Operations: insert, delete, modify

- each can violate one or more of the above constraints - how are these violations handled
- INSERT
    - provides a list of attribute values to be inserted as a new tuple
    - can violate domain constraints if values not in required domain
      ex: insert <055, volleyball, FP, -9.0, ...> into TOY
    - can violate key constraints if try to insert a key value already existing
      ex: insert <001, Joe Kennedy, ...> into CUSTOMER
    - can violate entity integrity constraint if key value is null
      ex: insert <null, 001, 325, ...> into ORDER
    - can violate referential integrity if foreign key value refers to a tuple that does not exist
      ex: insert <044, volleyball, GC, ...> into TOY
    - handle these violations by rejecting the insertion or allowing the user to attempt to fix the cause of the violation
- DELETE
    - removes a tuple from the relation
    - can violate referential integrity if a foreign key in a tuple in another relation contains the value of the primary key of the deleted tuple
      ex: delete MAN_ID = FP would violate ref int of TOY tuples with TOY_NUM = 011 and 221
    - fix it by rejecting the deletion or attempting to cascade the deletion (delete any tuple that refers to the deleted tuple) or modifying the remaining foreign key values (null it out)
- MODIFY
    - change the vlaues of one or more attributes in a relation
    - specify a conditio to select the tuple
    - if not modifying a primary key or foreign key - only domain constraints can be violated
    - if primary key modified- comes down to deleting one tuple and inserting another -see above for issues
    - if foreign key - referential integrity constraints can be violated

The Relational Algebra:

- a set of operations used to manipulate entire relations

SELECT - used to select a set of tuples in the relation using a selection condition

$\sigma_{\text{<selection-condition>}}(\text{<relation-name>})$

- selection condition is a Boolean expression made up of clauses of the form: <attrib-name> <comparison-operator> <attrib-name>, or
  <attrib-name> <comparison-operator> <constant-value>

- result is a relation with the same attributes as the one specified in relation-name

ex: $\sigma_{<MAN\_ID=FP>}$(<TOY>) = {(011,FARM HOUSE, FP, 29.95,...),
(221, EXERSAUCER, FP, 45.00,...)}

$\sigma_{<NUM\_CHILDREN > 5>}$(<CUSTOMER>) = { }

$\sigma_{<(DATE\_ORDERED < 07/30/96) AND (DATE\_DELIVERED = 00/00/00)>}$(<ORDER>)

- select conditions cannot apply to more than one relation at at time
- the fraction of tuples selected is the selectivity of the condition

PROJECT - used to select certain attributes of a relation

$\pi_{<attrib\text{-}list>}$(<relation-name>)

- resulting relation has only those attributes listed - and all tuples
    - degree equal to the number of attribs in list
- implicitly removes any duplicate tuples

ex: $\pi_{<TOY\_NUM, NUM\_IN\_STOCK>}$(<TOY>) = {(011,18), (302,50),(325,12),(221,254)}

- if the resulting relation has a key in the list, it has the same number of tuples as the original

- commutativity does not hold

Combinations of operations - two choices:
1) Write the desired request in one complex composite operation
2) Write each operation separately and name each intermediate resulting tuple

ex: Find the name of every toy between $10.00 and $30.00.

1) $\pi_{<NAME>}$($\sigma_{<(MSRP>=10.00) AND (MSRP<=30.00)>}$<TOY>)

2) MID_PRICE <- $\sigma_{<(MSRP>=10.00) AND (MSRP<=30.00)>}$<TOY>
$\pi_{<NAME>}$<MID_PRICE>

-renaming attributes may be necessary for complex operations:

NEW(TOY_NAME) <- $\pi_{<NAME>}$<MID_PRICE>

- new relation called NEW has one attribute called TOY_NAME

Set Operations:

UNION: $R_1(A_{11},A_{12},...,A_{1n})$ UNION $R_2(A_{21}, A_{22},..., A_{2n}) = \{t \mid t \text{ in } R_1$ or t in $R_2\}$

R1 and R2 must be union compatibile - degree A_1 = degree A_2 = n
- $dom(A_{1i}) = dom(A_{2i})$, for all i=1..n

- duplicates are eliminated

- ex: HIGH_PRICE <- $\sigma_{<(MSRP<10.00)}$ <TOY>
  MID_PRICE UNION HIGH_PRICE

- commutative and associative

INTERSECTION: R_1 INTERSECT $R_2 = \{t \mid t \text{ in } R_1$ AND t in $R_2\}$

- ex: FP_HIGH <- HIGH_PRICE INTERSECT
  $\sigma_{<(MAN\_ID=FP)}$ <TOY>
  - high priced toys made by Fischer Price

- commutative and associative

DIFFERENCE: R1 - R2 = $\{t \mid t \text{ in } R_1$ AND t not in $R_2\}$

- ex: TOY - FP_HIGH

  - all toys not high priced made by FP

- not commutative or associative

CARTESIAN PRODUCT: $R_1(A_{11},A_{12},...,A_{1n})$ X $R_2(A_{21}, A_{22},..., A_{2m})$ = $R_1(A_{11},A_{12},...,A_{1n},A_{21}, A_{22},..., A_{2m})$

- result has one tuple for every combination of tuples from R_1 and R_2 - # tuples = n*m

- ex: TOY X CUSTOMER :

(TOY_NUM, NAME, MAN_ID, MSRP, AGE_GROUP, NUM_IN_STOCK, SOLD_YTD, CUST_NUM, NAME, ADDRESS, NUM_CHILDREN, AGES, LAST_ORDER_DATE)
=
011 FARM HOUSE FP 29.95 18MOS-3YRS 18 2 011KAREN
  SMITH ...
011 FARM HOUSE FP 29.95 18MOS-3YRS 18 2 002 GEORGE
  GRANT ...

- problem: NAME appears twice - resolve this by changing attr names or by using alias - TOY.NAME, CUSTOMER.NAME

- cartesian product not used very often, but combined with select, becomes join operation

JOIN: used to combine related tuples from different relations - denoted |X|

- Result <- R_1 |X|$_{\text{<join-cond>}}$ R_2

    - <join-cond> = <cond> AND <cond> ... AND <cond>
      where each cond is A theta B
        - A attrib of R1
        - B attrib of R2
        - theta comparison operator
    - if theta is =, called equijoin

- ex: TOY |X|$_{\text{MAN\_ID=MAN\_ID}}$ MANUFACTURER
  yields the relation:

  011 FARM HOUSE FP 29.95 18mos-3yrs 18 2 FP
  FISCHER PRICE 111 MAIN St. LANSING, MI
  (999)888-2345
  302 TODDLER TABLE PS 45.00 18mos-3yrs 50 18 PS
  PLAYSCHOOL 902 E. WILEY RD, DENVER,
  CO (699)929-3949
  etc.

- notice that there are two fields representing the manufacturers id in the resulting relation
- use **natural join** to alleviate this problem - eliminates duplicate attributes - denoted by *
    - two join attributes that represent the same entity should have the same name - use rename if not

    - or use the following more general definition of natural join
    - Q <- R_1 *(list1, list2) R_2

        - list1 and list2 are attribute lists where each consecutive attrib in the lists are compared with equality in the natural join

Operational Completeness: the set of operators: {π, σ, UNION, -, X} is a complete set of relational operators
- that is, intersect and |X| can be expressed in terms of these operators

Non-relational operators - not in relational algebra, but useful

Aggregate functions - SUM, AVERAGE, MAX, etc.
- group tuples based on values of given attributes and perform computations on group

<grouping-attrib> $F_{\text{<function-list>}}$(<relation-name>)

- grouping-attrib - list of attributes
- function-list - list of <function> <attrib> pairs - perform function on attrib

- ex: RESULT(MAN_ID, AVERAGE_PRICE) <-
  MAN_ID $\mathcal{F}$ AVERAGE MSRP (TOY)

RESULT:

MAN_ID	AVERAGE_PRICE
FP	37.48
PS	45.00
FY	12.95

Recursive Closure - no recursive operation built in - must be done manually

OUTER JOIN: keep all tuples of one (or both) of the relations in the result

- left outer join]X| - keep all tuples of first relation and those that match from the second relation

  -ex: list all customers and any pending orders

  PENDING_ORDERS <- $\sigma_{<(DELIV=00/00/00)}$ ORDER
  CUST_ORDERS <-
  CUSTOMER]X| $_{CUST\_NUM=CUST\_NUM}$
  PENDING_ORDERS

  CUST_ORDERS:
  001 KAREN SMITH 42 KENMORE RD ...
  (NULL ORDER ATTRIBS)
  002 GEORGE GRANT 5 WOODFIELD ...
  5035 325 2 08/21/95 12 EASTGATE ...

- right outer join |X[- keep all tuples of second relation

OUTER UNION: take union of tuples from relations that are not union compatible
- attributes that are not union compatible from either relation are kept in the result
- tuples with no values for these attribs are padded with null values

Examples of Queries:

- look at examples in book using the Company database