Databases

Cap. 3. Relational Algebra. DB Normalization

Textbook: Ramakrishnan, Gehrke, “Database Management Systems”, McGraw Hill, 2003

2020 UPT Conf.Dr. Dan Pescaru

The Relational Model

1. Relational database: a collection formed by relations and links between them
2. Relation:
   - Schema: relation name, attributes (columns) and attributes domains (types)
   - Ex: Students(sid:String, name:String, year:Integer, grade:Real)
   - Instance: physical table, having a fixed number of columns and rows (tuples)
   - Number of columns = relation DEGREE (ARITY)
   - Number of rows = relation CARDINALITY
   - Note: Unordered rows (records)!
   - Ordered columns (ex. SQL INSERT)

Relation Instance

Students

sid	name	year	grade
AC2153	Pop Angela	2	8.50
AC1078	Avram Ioan	1	9.35
AC2056	Ionescu Mihai	2	7.80
AC3098	Georgescu Ana	3	9.00
AC3023	Mihu Andrei	3	6.30

- The Students relation
  - Degree: 4
  - Cardinality: 5
- Obs: *Contains only distinct rows*

Integrity constraints

1. A collection of logical statements (logical expressions) that must be satisfied by any instance of the database
2. Integrity constraints:
   - Specified when defining the database schema
   - Are checked at every change made in the database
3. A database instance that satisfies all integrity constraints is called legal (or "in a consistent state"). Verification of consistency is performed by the DBMS
4. Note: Checking constraints avoids some input errors

Keys

1. A set of attributes forms a key for a relation if:
   a. There is no pair of tuples that have equal values for all attributes in the set (ensure uniqueness)
   b. Any subset of it break the above property (must be minimal)
2. A superkey: a set of attributes that includes at least one key

The Primary Key (PK)

1. Primary key: a key chosen by the DB designer from the set of keys, which satisfies the following conditions:
   - Is short (comprising a minimum number of attributes - ideally just one)
     - Note: (unusual case) just one candidate key formed by all fields
   - Is representative in problem context (eg SID vs. SSN)
   - Do not allow NULL values
2. Alternative: introduction of an artificial key (usually numeric using AutoIncrement)

Referential Integrity

1. Foreign Key (FK) – links two tables (corresponds to the primary key of the main relation)
2. Used to check the referential integrity
   - When adding a record in the secondary table: the corresponding key should exists in the referenced table
   - When deleting a record from the referenced table: avoiding "orphan" records in the other table
   - When changing the value of a primary key or of a foreign key in the related tables

Ensuring referential integrity

1. For INSERT
   - block the addition of a record having a wrong reference (FK without corresponding PK)
2. For DELETE
   - Cascade delete related records
   - Avoiding operation that breaks constraints
   - Voiding the reference (unusual solution)
3. For UPDATE
   1. Cascade update related records
   2. Avoiding operation that breaks constraints
4. Issue: deadlocks (cross references), solution: using transactions (late checking)

Referential Integrity. Example

Students

sid	name	year	grade
AC2153	Pop Angela	2	8.50
AC1078	Avram Ioan	1	9.35
AC2056	Ionescu Mihai	2	7.80

Enrollments

eid	sid	lab	exam
PLA2	AC2153	9	8
UC1	AC1078	10	9
SO2	AC2056	8	7

1. INSERT a new Enrolment: verify the presence of sid in Students
2. DELETE a Student: verify sid in Enrolments
3. UPDATE a sid in Students: verify the presence of sid in Enrolments

Relational Algebra

1. Basic operators:
   - Projection (Π) – Delete unwanted columns from result
   - Selection (σ) – selects a subset of rows from a relation
   - Cartesian product ($\times$) – allows to combine two relations
   - Set difference ($/$) – tuples in R1, but not in R2
   - Union ($\cup$) – tuples in R1 and in R2 (discard duplicates)
2. Additional operations:
   - JOIN ($\bullet$), Intersection, Division, Renaming
3. Since each operation returns a relation, operations can be composed (Algebra is "closed")

Projection

1. Projection (Π) – Deletes attributes from input relation that are not in projection list
2. Schema of result contains exactly the fields in the projection list, with the same names that they had in the input relation
3. Projection operator has to eliminate duplicates! (*Why*??)
4. Note: real systems typically do not eliminate duplicates unless the user explicitly asks for it. (*Why not*?)
5. Ex: Π_{grade}(Students) = {10.0, 9.5, 8, 6.5}

Selection

1. Selection (σ) – selects rows that satisfy selection condition from an input relation
2. Selection condition: simple (comparing attributes) or complex (using logical connectives *AND*, *OR*, *NOT*)
3. Schema of result identical to schema of the input relation
4. Note: No duplicates in result! (*Why*?)
5. Result cardinality <= Input cardinality
6. Ex: $\sigma_{grade<9.0}$(Students)

Union, Intersection, Difference

1. All take two input relations, which must be union-compatible:
   - Have same number of fields
   - Corresponding field have the same type

S1

sid	name	year	grade
AC2034	Popescu Ion	2	9.25
AC2056	Ionescu Vasile	2	8.70

S2

sid	name	year	grade
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

sid	name	year	grade
AC2056	Ionescu Vasile	2	8.70

S1 ∩ S2

sid	name	year	garde
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

S2 / S1

sid	name	year	grade
AC2034	Popescu Ion	2	9.25
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

S1 ∪ S2

Cross Product

1. Each row of R1 is paired with each row of R2
2. Result schema has one field per field of R1 and R2, with field names *inherited* if possible (or generated using renaming operator)

Students

sid	name	year	grade
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

Cars

sid	cid	car_type
AC2056	TM06ABC	Dacia Logan
ET3045	TM01AAA	Renault Megane
ME1078	TM08BBB	Ford Focus
AC1012	TM09PPP	Opel Corsa

(sid)	name	year	grade	(sid)	cid	car_type
AC2056	Ionescu Vasile	2	8.70	AC2056	TM06ABC	Dacia Logan
AC2056	Ionescu Vasile	2	8.70	ET3045	TM01AAA	Renault Megane
AC2056	Ionescu Vasile	2	8.70	ME1078	TM08BBB	Ford Focus
AC2056	Ionescu Vasile	2	8.70	AC1012	TM09PPP	Opel Corsa
ET3045	Vasilescu Ana	3	9.00	AC2056	TM06ABC	Dacia Logan
ET3045	Vasilescu Ana	3	9.00	ET3045	TM01AAA	Renault Megane
...	...	...	...	...	...	...

SxM

Join

1. JOIN = selection on a join condition over a cross product
2. $R1 \circ_{Jcond} R2 = \sigma_{Jcond}(R1 \times R2)$

Students

sid	name	year	grade
AC2056	Ionescu Vasile	2	8.70
ET3045	Vasilescu Ana	3	9.00
ME1078	Pop Angela	1	8.25
AC1012	Ticu Gelu	1	9.50

Cars

sid	cid	car_type
AC2056	TM06ABC	Dacia Logan
ET3045	TM01AAA	Renault Megane
ME1078	TM08BBB	Ford Focus
AC1012	TM09PPP	Opel Corsa

(sid)	name	year	grade	(sid)	cid	car_type
AC2056	Ionescu Vasile	2	8.70	AC2056	TM06ABC	Dacia Logan
ET3045	Vasilescu Ana	3	9.00	ET3045	TM01AAA	Renault Megane
ME1078	Pop Angela	1	8.25	ME1078	TM08BBB	Ford Focus
AC1012	Ticu Gelu	1	9.50	AC1012	TM09PPP	Opel Corsa

Students $\circ_{Students.NRM=Cars.NRM}$ **Cars**

Join

1. Result schema same as that of cross-product, but fewer tuples. Might be able to compute more efficiently. How?
2. Types of JOIN:
   - Theta-join – general join condition (=,<,<>)
   - Equi-join – a special case of condition join where the condition contains only equalities
   - Natural-join – Equi-join on all common fields (default if no condition is specified) $\bowtie$
   - Self-join – R1 and R2 are the same

Example

Boat Reserves - DB Schema [1]

Sailors (S)

sid:int	sname:string	rating:int	age:real
22	Dustin	7	45.0
…			
58	John	10	52.0

Boats (B)

bid:int	bname:string	color:string
103	Clipper	green
…		
127	Neptune	blue

Reserves (R)

sid:int	bid:int	day:date
58	103	02/11/2010
…		
22	127	09/02/2013

[1] Ramakrishnan, Gehrke, “Database Management Systems”, McGraw Hill, 2003

Query: example

- Problem: find all sailors that reserves a green boat
- Solutions:

1. $\pi_{sid,sname}(\sigma_{color='green'}((S \circ_{S.sid=R.sid} R) \circ_{R.bid=B.bid} B))$

2. $\pi_{sid,sname}((S \circ_{S.sid=R.sid} R) \circ_{R.bid=B.bid}(\sigma_{color='green'} (B))$

- Two different execution plans
- Which one is the most efficient?

Relational model: the evils of redundancy

- Redundancy is at the root of several problems associated with relational schemas
  - Waste of storage space
  - Insert/delete/update anomalies
- Example: bad DB schema

Students

sid	name	year	grade	faculty	addr.
AC2153	Pop Angela	2	8.50	Automation and Computers	2 V. Parvan, Electro Building
AC1078	Avram Ioan	1	9.35	Automation and Computers	2 V. Parvan, Electro Building
AC2056	Ionescu Mihai	2	7.80	Automation and Computers	2 V. Parvan, Electro Building

Dealing with redundancy

- Integrity constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements
- Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD)
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Functional dependencies (FDs)

- A functional dependency $X \rightarrow Y$ holds over relation R if, for every allowable instance r of R:
- $t_1 \in R$, $t_2 \in R$, $\pi_X(t_1) = \pi_X(t_2) \Rightarrow \pi_y(t_1) = \pi_y(t_2)$

  (given two tuples in R, if the X values agree, then the Y values must also agree - X and Y are sets of attributes)

- An FD is a statement about all allowable relations
  - Must be identified based on semantics of enterprise
  - Given some allowable instance r1 of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!

Example: Constraints on Entity Set

- Consider a relation Hourly_Emps:

  Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)
- Notation: We will denote this relation schema by listing the attributes - SNLRWH
  - This is really the set of attributes {S,N,L,R,W,H}
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  - ssn is the key: S $\rightarrow$ SNLRWH
  - rating determines hrly_wages: R $\rightarrow$ W

Example: Constraints on Entity Set

- Problems due to R→W :
  - Update anomaly: can we change W in just the 1st tuple of SNLRWH?
  - Insertion anomaly: what if we want to insert an employee with a different hourly wage for his rating?
  - Deletion anomaly: if we delete all employees with rating 5, we lose the information about the wage for rating 5!

S	N	L	R	W	H
123-22-3666	Mark	48	8	10	40
231-31-5368	John	22	8	10	30
131-24-3650	Rolf	35	5	7	30
434-26-3751	Piere	35	5	7	32
612-67-4134	Gery	35	8	10	40

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - ssn $\rightarrow$ did $\wedge$ did $\rightarrow$ lot implies ssn $\rightarrow$ lot
- An FD f is implied by a set of FDs F if f holds whenever all FDs in F hold
  - F^+ (*closure* of F) is the set of all FDs that are implied by F
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - Reflexivity: if $X \subseteq Y$, then $Y \rightarrow X$
  - Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- These are *sound* and *complete* inference rules for FDs!

Example

- A relation R has attributes (S, C, T, R, G) which denotes student, course, time, room, and grade, respectively. From requirements, the following FDs hold:
  - SC $\rightarrow$ G
  - ST $\rightarrow$ R
  - C $\rightarrow$ T
  - TR $\rightarrow$ C

Attribute closure

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in number of attributes!)
- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F. An efficient check:
  - Compute attribute closure of X (denoted X^+):
    - Set of all attributes A such that $X \rightarrow A$ is in F^+
    - There is a linear time algorithm to compute this
  - Check if Y is in X^+

Attribute closure algorithm

- The algorithm for X^+ (closure) computation:

```
closure = X;
REPEAT UNTIL no change
{
   IF there is an FD U→V in F
         such that U ⊆ closure
   THEN
      SET closure = closure ∪ V
}
```

Normalization

1. Regarding database schema design, a good question to ask is whether any refinement is needed. Answer: in case of redundancy
2. Solution: normalization using decomposition
3. Reason: if a relation is in a certain normal form certain kinds of problems are avoided/ minimized
4. Using FDs in detecting redundancy: (R:ABC)
    1. No FDs hold: There is no redundancy here
    2. Given A $\rightarrow$ B: Several tuples could have the same A value, and if so, they will all have the same B value!

Normalization - history

1. **Edgar F. Codd** first proposed the process of normalization and what came to be known as the 1st normal form in his paper A Relational Model of Data for Large Shared Data Banks: "There is, in fact, a very simple elimination procedure which we shall call *normalization*. Through decomposition non-simple domains are replaced by 'domains whose elements are atomic (non-decomposable) values'."
2. He originally established three normal forms: **1NF**, **2NF** and **3NF**. There are now also others that are accepted, but 3NF is widely considered to be sufficient for most applications (DBs are considered normalized if they are in 3NF)

Relational scheme decomposition

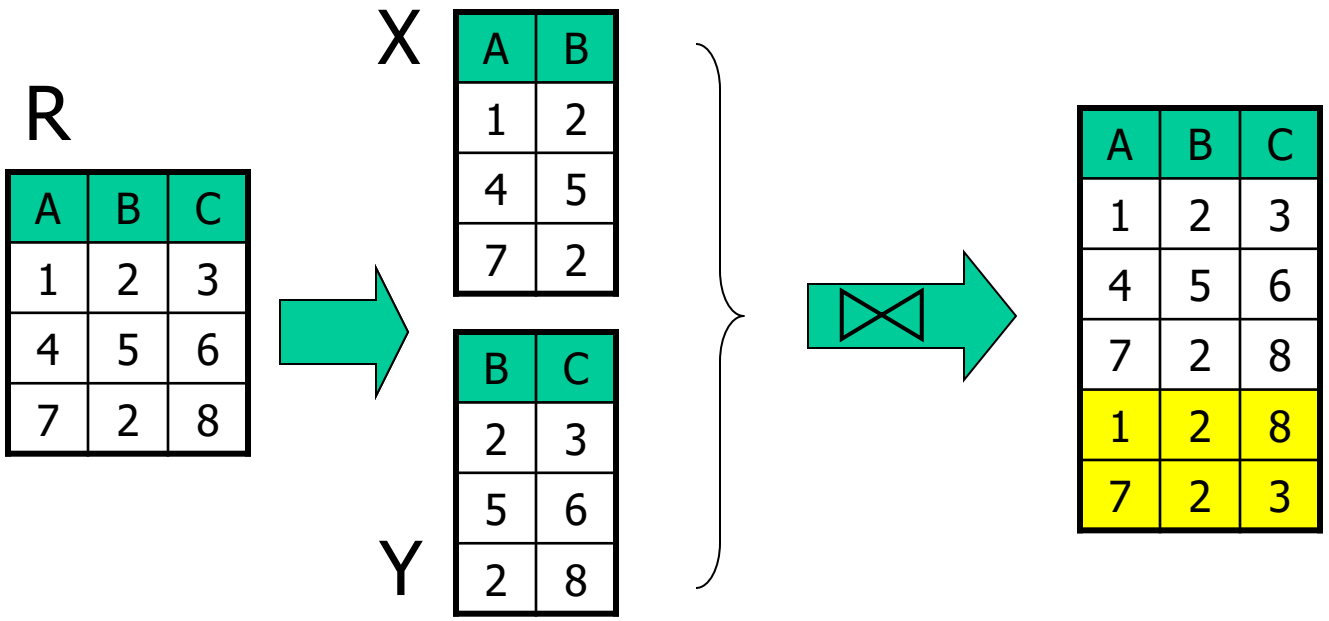
1. Suppose that relation R contains attributes A_1 ... A_n. A decomposition of R consists of replacing R by two or more relations such that:
   - Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R)
   - Every attribute of R appears as an attribute of one of the new relations
2. Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R
3. E.g., Can decompose SNLRWH into SNLRH and RW

Lossless join decomposition

1. Decomposition of R into X and Y is lossless-join with respect to a set of FDs F if, for every instance r that satisfies F:
   - $\Pi_X(r) \bowtie \Pi_Y(r) = r$
2. Definition extended to decomposition into 3 or more relations in a straightforward way
3. *It is essential that all decompositions used to deal with redundancy be lossless* !

Lossless join example

1. The following decomposition of R into X and Y has not the lossless-join property!



Dependencies preservation

1. Decomposition must keep all dependencies of the original relation in order to keep all constraints!
   - Condition: $(F1 \cup F2)^+ = F^+$
2. Example

R=(A,B,C), F={A→B, A→C, B→C}

R1=(A,C), R2(B,C)

$F1^+$={A→A, C→C, A→C, AC→AC}

$F2^+$={B→B, C→C, B→C, BC→BC}

- Note: A→B <u>is not preserved</u>

First normal form (1NF)

1. First normal form (1NF):
   - The domain of each attribute must contain only atomic values; composite fields or "relationships into relationship" are prohibited
   - Each attribute contains only a single value from that domain

Students

sid	name	hobby	fid	address
AC6978	Popescu Mihai	chess, dance	AC	Timisoara, Parvan **no 3**, 0256112212
AC8967	Ionescu Georgeta	reading, painting	AC	Timisoara, Parvan **no 1**, 0256112212

1NF example

1. Solution: two attribute split and one decomposition

Students

sid	name	hobby	fid	address
AC6978	Popescu Mihai	chess, dance	AC	Timisoara, Parvan no 3, 0256112212
AC8967	Ionescu Georgeta	reading, painting	AC	Timisoara, Parvan no 1, 0256112212

Students

sid	fname	sname	Fid	Addr_town	Addr_str	phone
AC6978	Mihai	Popescu	AC	Timisoara	2 V. Parvan	0256112212
AC8967	Georgeta	Ionescu	AC	Timisoara	1 Victoriei Sq.	0256334434

Hobbies

sid	hobby
AC6978	chess
AC6978	dance
AC8967	reading

Second normal form (2NF)

1. Second normal form (2NF):
   - Relation is already in 1NF
   - Any non-prime attribute of R (not part of the primary key) must be fully functionally dependent on the primary key of R
   - OR: there are no attributes that depend only on a part of the primary key

Invoice

id	date	poz	name	price
008978	01-03-2014	1	bread	2
008978	01-03-2014	2	apples	6
099488	05-03-2014	1	cheese	17

2NF Example

1. Solution: decomposition

Invoice

id	date
008978	01-03-2014
008978	01-03-2014
099488	05-03-2014

Items

id	poz	name	price
008978	1	bread	2
008978	2	apples	6
099488	1	cheese	17

Third normal form (3NF)

1. Third normal form (3NF):
   - Relation is already in 2NF
   - Every non-prime attribute is non-transitively dependent on every candidate key in the table. In other words, no transitive dependency is allowed
2. BCNF: Every non-trivial functional dependency in the table is a dependency on a superkey

Items

id	poz	name	quantity	unit_price	price
008978	1	bread	2	2	4
008978	2	apples	5	6	30
099488	1	cheese	1	17	17

3NF Example

1. Solution: decomposition

Items

id	poz	name	quantity	unit_price	price
008978	1	bread	2	2	4
008978	2	apples	5	6	30
099488	1	cheese	1	17	17

Product_catalog

name	unit_price
bread	2
apples	6
cheese	17

Fourth normal form (4NF)

1. Fourth normal form (4NF):
   - Relation is already in 3NF
   - There are no multivalued dependencies
2. Multivalued Dependencies – considering the schema ABC, multivalued dependency exists if to each A corresponds many B and many C, but the B and C are independent of each other

Components

Department	Project	Component
D1	Pr1 Pr2	C1 C2 C3
D2	Pr2 Pr3 Pr5	C2 C4

Components (3NF)

Department	Project	Component
D1	Pr1	C1
D1	Pr1	C2
D1	Pr1	C3
D1	Pr2	C1
D2	Pr2	C2
…	…	…
D2	Pr5	C4

4NF Example

1. Problem: lets suppose that departments maintain a stock of components and they develop projects that can use some of the available components or all. Solution: decomposition

Components (3NF)

Department	Project	Component
D1	Pr1	C1
D1	Pr1	C2
D1	Pr1	C3
D1	Pr2	C1
D2	Pr2	C2
…	…	…
D2	Pr5	C4

DP (4NF)

Department	Project
D1	Pr1
D1	Pr2
D2	Pr2
…	…

DC (4NF)

Department	Component
D1	C1
D1	C2
D1	C3
D2	C2
…	…

Relational query languages

1. Relational query language: designed for data finding, retrieving and management
2. Relational model enable simple and powerful query languages (e.g. SQL, QBE):
   - Strong formal foundation based on algebra/logic
   - Allows automatic optimization
3. Query languages are not programming languages:
   1. Pure QL are not Turing-complete (SQL92), but extensions as Oracle PL/SQL could be
   2. E.g. Recursive closure could not be expressed
4. But they provide efficient data access!