Database Management Systems INFO 210

Summary - Part 2 (SQL,Relational Algebra, NFs)

Franz Wotawa
wotawa@ist.tugraz.at

1

Relational Query Languages

- Query languages (QLs) allow *manipulating* and *retrieving* data from databases

- The relational model supports simple and powerful QLs:
  - Strong formal foundation based on logic
  - High amenability for effective optimizations

- Query Languages != programming languages!
  - QLs are not expected to be "Turing complete"
  - QLs are not intended to be used for complex calculations
  - QLs support easy and efficient access to large datasets

2

SQL

3

DDL and DML

- The SQL language has two main aspects (*there are other aspects which we will discuss in next classes*)
  - Data Definition Language (DDL)
    - Allows creating, modifying, and deleting relations and views
    - Allows specifying constraints
    - Allows administering users, security, etc.
  - Data Manipulation Language (DML)
    - Allows posing *queries* to find tuples that satisfy criteria
    - Allows adding, modifying, and removing tuples

4

Creating Relations in SQL

- S1 can be used to create the "Students" relation
- S2 can be used to create the "Enrolled" relation

```
CREATE TABLE Students
        (sid: CHAR(20),
         name: CHAR(20),
         login: CHAR(10),
         age: INTEGER,
         gpa: REAL)
```

S1

```
CREATE TABLE Enrolled
        (sid: CHAR(20),
         cid: CHAR(20),
         grade: CHAR(2))
```

S2

The DBMS enforces domain constraints whenever tuples are added or modified

5

Adding and Deleting Tuples

- We can insert a single tuple to the "Students" relation using:

```
INSERT INTO  Students (sid, name, login, age, gpa)
VALUES  (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- We can delete all tuples from the "Students" relation which satisfy some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

Powerful variants of these commands are available!

6

Querying a Relation

- How can we find all 18-year old students?

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- How can we find just names and logins?

```
SELECT S.name, S.login
FROM Students S
WHERE S.age=18
```

7

Querying Multiple Relations

- What does the following query compute assuming S and E?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

S

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

E

We get:

S.name	E.cid
Smith	Topology112

8

Destroying and Altering Relations

- How to destroy the relation "Students"?

```
DROP TABLE Students
```

The schema information *and* the tuples are deleted

- How to alter the schema of "Students" in order to add a new field?

```
ALTER TABLE Students
        ADD COLUMN firstYear: integer
```

Every tuple in the current instance is extended with a ***null*** value in the new field!

9

Integrity Constraints (ICs)

- An IC is a condition that must be true for *any* instance of the database (e.g., *domain constraints*)
  - ICs are specified when schemas are defined
  - ICs are *checked* when relations are modified
- A *legal* instance of a relation is one that satisfies all specified ICs
  - DBMS should not allow illegal instances
- If the DBMS checks ICs, stored data is more faithful to real-world meaning
  - Avoids data entry errors, too!

10

Keys

- Keys help associate tuples in different relations
- Keys are one form of integrity constraints (ICs)

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

11

Keys

- Keys help associate tuples in different relations
- Keys are one form of integrity constraints (ICs)

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

FOREIGN Key

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

PRIMARY Key

12

Superkey, Primary and Candidate Keys

- A set of fields is a *superkey* if:
  - No two distinct tuples can have same values in *all* key fields
- A set of fields is a *primary key* for a relation if:
  - It is a *minimal* superkey
- What if there is more than one key for a relation?
  - One of the keys is chosen (by DBA) to be the primary key
  - Other keys are called *candidate keys*
- Examples:
  - *sid* is a key for Students (what about *name*?)
  - The set {*sid, name*} is a superkey (or a set of fields that contains a key)

13

Primary and Candidate Keys in SQL

- Many candidate keys (specified using UNIQUE) can be designated and one is chosen as a *primary key*
- Keys must be used carefully!
- "For a given student and course, there is a single grade"

14

Primary and Candidate Keys in SQL

- Many candidate keys (specified using UNIQUE) can be designated and one is chosen as a *primary key*
- Keys must be used carefully!
- "For a given student and course, there is a single grade"

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))
```

VS.

```
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid),
    UNIQUE (cid, grade))
```

15

Primary and Candidate Keys in SQL

- Many candidate keys (specified using UNIQUE) can be designated and one is chosen as a *primary key*
- Keys must be used carefully!
- "For a given student and course, there is a single grade"

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))
```

VS.

```
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid),
    UNIQUE (cid, grade))
```


Q: What does this mean?

16

Primary and Candidate Keys in SQL

- Many candidate keys (specified using UNIQUE) can be designated and one is chosen as a *primary key*
- Keys must be used carefully!
- "For a given student and course, there is a single grade"

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))
```

vs.

```
CREATE TABLE Enrolled
    (sid CHAR(20)
     cid  CHAR(20),
     grade CHAR(2),
     PRIMARY KEY  (sid),
     UNIQUE (cid, grade))
```

"A student can take only one course, and no two students in a course receive the same grade"

17

Foreign Keys and Referential Integrity

- A foreign key is a set of fields referring to a tuple in another relation
  - It must correspond to the primary key of the other relation
  - It acts like a `logical pointer'
- If all foreign key constraints are enforced, referential integrity is said to be achieved (i.e., no dangling references)

18

Foreign Keys in SQL

- Example: Only existing students may enroll for courses
  - *sid* is a foreign key referring to Students
  - How can we write this in SQL?

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

19

Foreign Keys in SQL

- Example: Only existing students may enroll for courses

```
CREATE TABLE Enrolled
(sid CHAR(20),cid CHAR(20),grade CHAR(2),
 PRIMARY KEY (sid,cid),
 FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

20

Enforcing Referential Integrity

- What should be done if an "Enrolled" tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a "Students" tuple is deleted?
  - Disallow its deletion
  - Delete all Enrolled tuples that refer to it
  - Set sid in Enrolled tuples that refer to it to a *default sid*
  - Set sid in Enrolled tuples that refer to it to a special value *null*, denoting *`unknown'* or *`inapplicable'*
- What if a "Students" tuple is updated?

21

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates
  - Default is NO ACTION (i.e., *delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to the deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid)
    REFERENCES Students
          ON DELETE CASCADE
          ON UPDATE SET DEFAULT )
```

What does this mean?

22

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance
  - An IC is a statement about *all possible* instances!
  - From the “Students” relation, we know *name* is not a key, but the assertion that *sid* is a key is given to us
- Key and foreign key ICs are the most common; more general ICs are supported too

23

Views

- A view is a table whose rows are not explicitly stored but computed as needed

```
CREATE VIEW YoungActiveStudents (name, grade)
    AS SELECT S.name, E.grade
    FROM Students S, Enrolled E
    WHERE S.sid = E.sid and S.age<21
```

- Views can be queried
  - Querying YoungActiveStudents would necessitate computing it first then applying the query on the result as being like any other relation
- Views can be dropped using the DROP VIEW command
  - How to handle DROP TABLE if there’s a view on the table?
    - DROP TABLE command has options to let the user specify this

24

Formal Relational Query Languages

- There are two mathematical Query Languages which form the basis for commercial languages (e.g. SQL)
  - Relational Algebra
    - Queries are composed of operators
    - Each query describes a step-by-step procedure for computing the desired answer
    - Very useful for representing *execution plans*
  - Relational Calculus
    - Queries are subsets of first-order logic
    - Queries describe desired answers without specifying how they will be computed
    - A type of *non-procedural* (or *declarative*) formal query language

25

Relational Algebra

- Operators (with notations):
  1. Selection (σ)
  2. Projection (π)
  3. Cross-product (X)
  4. Set-difference (-)
  5. Union (∪)
  6. Intersection (∩)
  7. Join (⋈)
  8. Division (÷)
  9. Renaming (ρ)

- Each operation returns a relation, hence, operations can be *composed*! (i.e., Algebra is "closed")

26

Relational Algebra

- Operators (with notations):
  1. Selection (σ)
  2. Projection (π)
  3. Cross-product ($\times$)
  4. Set-difference (—)
  5. Union ($\cup$)
  6. Intersection ($\cap$)
  7. Join ($\bowtie$)
  8. Division ($/$)
  9. Renaming (ρ)

Basic

Additional, yet extremely useful!

- Each operation returns a relation, hence, operations can be *composed*! (i.e., Algebra is "closed")

27

The Projection Operation

- Projection: $\pi_{att-list}(R)$
  - "Project out" attributes that are NOT in *att-list*
  - The schema of the output relation contains ONLY the fields in att-list, with the same names that they had in the input relation

- Example 1: $\pi_{sname,rating}(S2)$

Input Relation:

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

Output Relation:

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

28

The Selection Operation

- Selection: $\sigma_{condition}(R)$
  - Selects rows that satisfy the selection *condition*
  - The schema of the output relation is identical to the schema of the input relation

- Example: $\sigma_{rating>8}(S2)$

Input Relation:

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

Output Relation:

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

29

Operator Composition

- *The output* relation can be the *input* for another relational algebra operation! (*Operator composition*)

- Example: $\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Input Relation:

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

Intermediate Relation:

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

Final Output Relation:

sname	rating
yuppy	9
rusty	10

30

The Union Operation

- Union: **R U S**
  - The two input relations must be union-compatible
    - Same number of fields
    - \`Corresponding' fields have the same type
  - The output relation includes all tuples that occur "in either" R or S "or both"
  - The schema of the output relation is identical to the schema of R

- Example: $S1 \cup S2$

Input Relations:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

Output Relation:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

31

The Intersection Operation

- Intersection: $R \cap S$
  - The two input relations must be *union-compatible*
  - The output relation includes all tuples that occur "in both" R and S
  - The schema of the output relation is identical to the schema of R

- Example: $S1 \cap S2$

Input Relations:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

Output Relation:

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

32

The Set-Difference Operation

- Set-Difference: $\boldsymbol{R - S}$
  - The two input relations must be *union-compatible*
  - The output relation includes all tuples that occur in R "but not" in S
  - The schema of the output relation is identical to the schema of R
- Example: $S1 - S2$

Input Relations:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

Output Relation:

sid	sname	rating	age
22	dustin	7	45.0

33

The Cross-Product and Renaming Operations

- Cross Product: $\boldsymbol{RXS}$
  - Each row of R is paired with each row of S
  - The schema of the output relation concatenates S1's and R1's schemas
  - Conflict: R and S might have the same field name
  - Solution: Rename fields using the "Renaming Operator"
  - Renaming: $\rho(R(\overline{F}), E)$
- Example: $S1XR1$

Input Relations:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1

Output Relation:

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Conflict: Both S1 and R1 have a field called *sid*

34

The Cross-Product and Renaming Operations

- Cross Product: $\boldsymbol{RXS}$
  - Each row of R is paired with each row of S
  - The schema of the output relation concatenates S1's and R1's schemas
  - Conflict: R and S might have the same field name
  - Solution: Rename fields using the "Renaming Operator"
  - Renaming: $\rho(R(\overline{F}), E)$
- Example: $S1XR1$

Input Relations:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1

Output Relation:

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

$$\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$$

35

The Join Operation

- (Theta) Join : $R \bowtie_c S = \sigma_c(R \times S)$
  - The schema of the output relation is the same as that of cross-product
  - It usually includes fewer tuples than cross-product
- Example: $S1 \bowtie_{S1.sid < R1.sid} R1$

Input Relations:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1

Output Relation:

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

Will be redundant "if" the condition is S1.sid = R1.sid!

36

The Join Operation

- Equi-Join: $R \bowtie_c S = \sigma_c(R \times S)$
  - A special case of theta join where the condition *c* contains only ***equalities***
  - The schema of the output relation is the same as that of cross-product, *"but only one copy of the fields for which equality is specified"*
- Natural Join: $R \bowtie S$
  - Equijoin on *"all"* common fields
- Example: $S1 \bowtie R1$ ← Natural Join $S1 \bowtie_{S1.sid=R1.sid} R1$

Input Relations:

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Output Relation:

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

ONLY one sid column!

37

The Division Operation

- Division: $R \div S$
  - Not supported as a primitive operator, but useful for expressing queries like:

    Find sailors who have reserved all boats
  - Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:
    - *A/B* contains all *x* tuples (sailors) such that for *every* *y* tuple (boat) in *B*, there is an *xy* tuple in *A*
    - *Or*: If the set of *y* values (boats) associated with an *x* value (sailor) in *A* contains all *y* values in *B*, then *x* value is in *A/B*
    - Formally: A/B = $\{\langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B\}$
  - In general, *x* and *y* can be any lists of fields; *y* is the list of fields in *B*, and *x* *y* is the list of fields in *A*

38

NORMAL FORMS

39

Example of data inconsistencies

- Change of the address of a lecture hall!

Lect_num	Type	Name	Hours	Teacher	Room	Address
117.101	VO	Software Maintenance	3.0	Franz Wotawa	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	Franz Wotawa	LH i11	Infeldgasse 16
117.111	UE	Compiler Construction	1.0	Birgit Hofer	LH i11	Inffeldgasse 16
....	......	....	.....	....	....	

Change 16 to 17!

Lect_num	Type	Name	Hours	Teacher	Room	Address
117.101	VO	Software Maintenance	3.0	Franz Wotawa	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	Franz Wotawa	LH i11	Infeldgasse 17
117.111	UE	Compiler Construction	1.0	Birgit Hofer	LH i11	Inffeldgasse 16
....	......	....	.....	....	....	

≠

40

1. Normal form (1NF)

- *"A relation is in first normal form if and only if the* ***domain*** *of each attribute* ***contains only atomic (indivisible) values****, and the value of each attribute contains only a single value from that domain."*
- The 1NF enables queries and sorting!
- There is one extension: *"... and each relation must have a primary key."*

41

Example 1NF

Lect_num	Type	Name	Hours	Teacher	Room	Address
117.101	VO	Software Maintenance	3.0	Franz Wotawa, Birgit Hofer	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	Franz Wotawa	LH i11	Infeldgasse 16
117.111	UE	Compiler Construction	1.0	Birgit Hofer	LH i11	Inffeldgasse 16
....	......	....	.....	....	....	

The relation above is not in 1NF!!!

Lect_num	Type	Name	Hours	Teacher	Room	Address
117.101	VO	Software Maintenance	3.0	Franz Wotawa	LH i7	Inffeldgasse 25
117.101	VO	Software Maintenance	3.0	Birgit Hofer	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	Franz Wotawa	LH i11	Infeldgasse 16
117.111	UE	Compiler Construction	1.0	Birgit Hofer	LH i11	Inffeldgasse 16
....	......	....	.....	....	....	

42

2. Normal form (2NF)

- *"A relation in 1NF is in 2NF if and only if no* ***non-prime attribute*** *is* ***dependent*** *on any* ***proper subset*** *of any (candidate) key of the relation.*

  A non-prime attribute *of a relation is an attribute that is* ***not part of any*** *(candidate)* ***key*** *of the relation."*

  Every non-prime attribute has to be dependent on the key only

- A database that is not in 2NF comprises redundancies!

43

Example 2NF

- Let us consider the following table with primary key {Lect_num,SSN}:

Lect_num	Type	Name	Hours	SSN	Teacher	Room	Address
117.101	VO	Software Maintenance	3.0	1000	Franz Wotawa	LH i7	Inffeldgasse 25
117.101	VO	Software Maintenance	3.0	1002	Birgit Hofer	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	1000	Franz Wotawa	LH i11	Infeldgasse 16
117.111	UE	Compiler Construction	1.0	1002	Birgit Hofer	LH i11	Inffeldgasse 16
....	......	....	.....		....	....	

- Dependencies:
  - {Lect_num} →*{Type, Name, Hours, Room}*
  - {SSN} →*{Teacher}*
  - *{Room}* →*{Address}*

44

Example 2NF

This table is not in 2NF!!!

- Type, name, and hours of a lecture do only depend on the lecture number (Lect_num) and not on the teacher!

Redundancy

Lect_num	Type	Name	Hours	SSN	Teacher	Room	Address
117.101	VO	Software Maintenance	3.0	1000	Franz Wotawa	LH i7	Inffeldgasse 25
117.101	VO	Software Maintenance	3.0	1002	Birgit Hofer	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	1000	Franz Wotawa	LH i11	Infeldgasse 16
117.111	UE	Compiler Construction	1.0	1002	Birgit Hofer	LH i11	Inffeldgasse 16
....	......	....			....	....	

Make 3 tables!!!

Lect_num	Type	Name	Hours	Room	Address
117.101	VO	Software Maintenance	3.0	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	LH i11	Infeldgasse 16
117.111	UE	Compiler Construction	1.0	LH i11	Inffeldgasse 16
....	......	....	.....	....	

Lect_num	SSN
117.101	1000
117.101	1002
117.102	1000
117.111	1002
....	....

SSN	Teacher
1000	Franz Wotawa
1002	Birgit Hofer
....	....

45

3. Normal form (3NF)

- *"A relation in 2NF is in 3NF if and only if all the attributes in a table are determined only by the candidate keys of that relation and not by any non-prime attributes.*

  No non-prime attributes are allowed to be transitive dependent on a prime attribute*!"*

- Eliminates problems occurring when changing information!

46

Example 3NF

{Lect_num} → {*Type, Name, Hours, Room*}
{*Room*} → {*Address*}

- **This table is not in 3NF!!!**
  - The room of the lecture depends functionally on the lecture number (Lect_num). The address of the room depends functionally on the room itself. Hence, we have a transitive dependency between a key and an attribute that is not in the key!

Lect_num	Type	Name	Hours	Room	Address
117.101	VO	Software Maintenance	3.0	LH i7	Inffeldgasse 25
117.101	VO	Software Maintenance	3.0	LH i7	Inffeldgasse 25
117.102	VO	Compiler Construction	2.0	LH i11	Infeldgasse 16
117.111	UE	Compiler Construction	1.0	LH i11	Inffeldgasse 16
....	......	....	.....	....	

Make 2 tables!!!

Lect_num	Type	Name	Hours	Room
117.101	VO	Software Maintenance	3.0	LH i7
117.101	VO	Software Maintenance	3.0	LH i7
117.102	VO	Compiler Construction	2.0	LH i11
117.111	UE	Compiler Construction	1.0	LH i11
....	......	....	.....	....

Room	Address
LH i7	Inffeldgasse 25
LH i7	Inffeldgasse 25
LH i11	Infeldgasse 16
LH i11	Inffeldgasse 16
....	

47

Boyce-Codd Normal Form

A relation *R* is in Boyce-Codd Normal Form (*BCNF*) if whenever $X \rightarrow A$ is a *nontrivial* FD that holds in *R*, then *X* is a *superkey*

Remember:

- *nontrivial* means $A \notin X$
- a *superkey* is any superset of a key (not necessarily a strict superset)

"Each attribute must describe the key, the whole key, and nothing but the key"

48

All two-attribute relations are in BCNF

R(A,B)

Case 1: Suppose there is no dependency

Nothing is being violated, so fine

Case 2: Suppose A → B is the only dependency

A^+ = AB, so left side of A → B is a key

Case 3: Suppose B → A is the only dependency

B^+ = AB, so left side of B → A is a key

Case 4: Suppose A→ B and B → A are two dependencies

A^+ = AB, B^+ = AB,

so left side of B → A is a key and left side of A→B is also a key

49

What 3NF and BCNF Give You

There are two important properties of a decomposition:

- *Losslessness*: It should be possible
  - to project the original relation onto the decomposed schema
  - and then reconstruct the original by a natural join
- *Dependency Preservation*: It should be possible to check in the projected relations whether all the given FDs are satisfied

5

50

Functional Dependencies

- A form of constraint
  - hence, part of the schema
- Finding them is part of the database design
- Also used in normalizing the relations

51

Functional Dependencies

Definition:

If two tuples agree on the attributes

$\boxed{A_1, A_2, \ldots, A_n}$

then they must also agree on the attributes

$\boxed{B_1, B_2, \ldots, B_m}$

Formally:

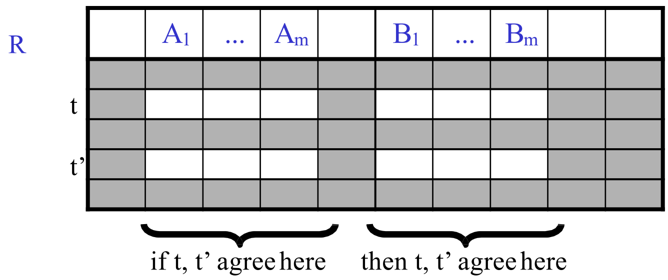
$\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$

52

When Does a FD Hold

Definition: $A_1, ..., A_m \rightarrow B_1, ..., B_n$ holds in R if:

$\forall t, t' \in R, (t.A_1=t'.A_1 \wedge ... \wedge t.A_m=t'.A_m \Rightarrow t.B_1=t'.B_1 \wedge ... \wedge t.B_n=t'.B_n)$



53

Example: Movie table

Title	Year	Length	Genre	StudioName	StarName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	Drama	MGM	Vivien Leigh
Wayne's World	1992	95	Comedy	Paramount	Dana Carvey
Wayne's World	1992	95	Comedy	Paramount	Mike Meyers

title, year → length

title, year → genre

title, year → studioName

title, year → length, genre, studioName

54

Example: Movie table

Title	Year	Length	Genre	StudioName	StarName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	Drama	MGM	Vivien Leigh
Wayne's World	1992	95	Comedy	Paramount	Dana Carvey
Wayne's World	1992	95	Comedy	Paramount	Mike Meyers

How about?

~~title, year → starName~~

55

Examples

An FD holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E11 1	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID → Name, Phone, Position

but not Name → EmpID

or Name → Phone

56

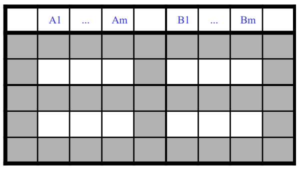
Armstrong's Rules (1/3)

$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Is equivalent to

$A_1, A_2, \ldots, A_n \rightarrow B_1$
$A_1, A_2, \ldots, A_n \rightarrow B_2$
. . . . .
$A_1, A_2, \ldots, A_n \rightarrow B_m$

Splitting rule and Combing rule



57

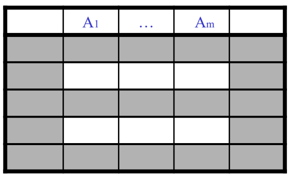
Armstrong's Rules (2/3)

$A_1, A_2, \ldots, A_n \rightarrow A_i$

Trivial Rule

where i = 1, 2, ..., n

Why ?



58

Armstrong's Rules (3/3)

Transitive Closure Rule

If $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

and $B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p$

then $A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p$

Why ?

59

	A_1	…	A_m		B_1	…	B_m		C_1	...	C_p	

60

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs

The **closure**, $\{A_1, \ldots, A_n\}^+$ = the set of attributes B
s.t. $A_1, \ldots, A_n \rightarrow B$

61

Closures Example

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office-supp.	59

Example:

name → color
category → department
color, category → price

Closures:

name$^+$ = {name, color}
{name, category}$^+$ = {name, category, color, department, price}
color$^+$ = {color}

62

Closure Algorithm

X={A1, …,An}.

Repeat until X doesn't change **do**:

if $B_1, \ldots, B_n \rightarrow C$ is a FD **and** $B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

name → color
category → department
color, category → price

{name, category}$^+$ =
{ name, category, color, department, price }

Hence: name, category → color, department, price

63

Does a new FD logically follows from a set of FDs?

1

$R(A_1, A_2, \ldots, A_m)$

Set of FDs ⟹ A new FD

- To check if $X \rightarrow A$ (new FD)
  - Using given set of FDs Compute X^+ i.e., {left side}$^+$
  - Check if $A \in X^+$

64

2

Using Closure to Infer ALL FDs

Example:

A, B $\rightarrow$ C
A, D $\rightarrow$ B
B $\rightarrow$ D

Step 1: Compute X^+, for every $X \subseteq \{A,B,C,D\}$:

A+ = A, B+ = BD, C+ = C, D+ = D
AB+ =ABCD, AC+=AC, AD+=ABCD,
BC+=BCD, BD+=BD, CD+=CD
ABC+ = ABD+ = ACD^+ = ABCD (no need to compute– why?)
BCD^+ = BCD, ABCD+ = ABCD

Step 2: Enumerate all FD's X $\rightarrow$ Y, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$

AB $\rightarrow$ CD, AD$\rightarrow$BC, BC $\rightarrow$D, ABC $\rightarrow$ D, ABD $\rightarrow$ C, ACD $\rightarrow$ B

65

3

Application of Closure: Finding Keys

- A **superkey** is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute B, we have $A_1, ..., A_n \rightarrow B$

- A **key** is a minimal superkey
  - i.e. set of attributes which is a superkey and for which no subset is a superkey

66

Computing (Super)Keys

- Compute X^+ for all sets X
- If X^+ = all attributes, then X is a key
- List only the minimal X's

67

4 Decomposition into BCNF

Given: relation R with FDs *F*

Goal: decompose R into relations $R_1,\ldots,R_m$ such that
- each R_i is a *projection* of R
- each R_i is in *BCNF*
  (wrt the projection of *F*)
- R is the *natural join* of $R_1,\ldots,R_m$

Intuition: R is broken into pieces
- that contain the same information as R,
- but are free of redundancy

68

The Algorithm: Divide and Conquer

- Look in F for an FD $X \rightarrow B$ that violates BCNF
  (If any FD following from F violates BCNF, then there is surely an FD in F itself that violates BCNF)
- Compute X^+
  (X^+ does not contain all attributes of R, otherwise X would be superkey)
- Decompose R using $X \rightarrow B$, i.e., replace R by relations with schemas
  $R_1 = X^+$
  $R_2 = (R - X^+) \cup X$
- Compute the projections F_1, F_2 of F on R_1, R_2
- Continue with R_1, F_1 and R_2, F_2

6

69

BCNF decomposition example

R(A,B,C,D,E)

A → C
B → D
C , B → E

Is this relation in BCNF?
If not, convert it into BCNF.
Is the conversion dependency preserving?

70