CS2102

RELATIONAL ALGEBRA

- Relations are closed under the Relational Algebra

Unary operators

Selection σ_c

- For each tuple $t \in R$, $t \in \sigma_c(R) \iff$ selection condition c evaluates to true for tuple t.
- Input and output have same schema
- e.g. Find all projects where Judy is the manager:
$$\sigma_{\text{manager}=\text{'Judy'}}(\text{Projects})$$

Selection condition is a boolean expression of one of the following forms:

expression	example
attribute **op** constant	$\sigma_{\text{start}=2020}(\text{Projects})$
$attr_1$ **op** $attr_2$	$\sigma_{\text{start}=\text{end}}(\text{Projects})$
$expr_1 \wedge expr_2$	$\sigma_{\text{start}=2020 \wedge \text{end}=2021}(\text{Projects})$
$expr_1 \vee expr_2$	$\sigma_{\text{start}=2020 \vee \text{end}=2021}(\text{Projects})$
$\neg\, expr$	$\sigma_{\neg(\text{start}=2020)}(\text{Projects})$
$(expr)$	-

where

- **op** $\in \{=, <>, <, \leq, \geq, >\}$
- Precedence: $()$, **op**, $\neg, \wedge, \vee$
- Comparision with **null** is **unknown**, arithmetic with **null** is **null**

In boolean expressions, treat unknown as literally unknown. e.g.

- false $\wedge$ unknown = false
- false $\vee$ unknown = unknown
- $\neg$ unknown = unknown
- true $\wedge$ unknown = unknown
- true $\vee$ unknown = true

Projection π_l

- Projects columns of a table specified in list l
- Order of attributes in l matters
- Duplicates are removed, because a relation is a set of tuples

Example

Teams

en	pn	hours
Sarah	BigAI	10
Sam	BigAI	5
Sam	BigAI	3

$\pi_{\text{pn,en}}(\text{Teams})$

pn	en
BigAI	Sarah
BigAI	Sam

Renaming ρ_l

- Renames attributes of a relation

Consider R(ename, pname, hours). Rename ename to name, pname to title. Can either specify

- list of all attr.: $\rho_{(\text{name, title, hours})}(R)$
- or list of renames:
$$\rho_{\text{name} \leftarrow \text{ename, title} \leftarrow \text{pname}}(R)$$

Set operations

- Union, Intersection, Set difference (all obvious)
- Note: intersection can be expressed with union and set difference:
$$R \cap S = (R \cup S) - ((R - S) \cup (S - R))$$
- The two relations must be union-compatible

Union compatability

Two relations are union-compatible if

- Same number of attributes
- Corresponding attributes have same or compatible domains (different attribute names are ok)

Example The following are union-compatible.

- Employees(name: **text**, role: **text**, age: **integer**)
- Teams(ename: **text**, pname: **text**, hours: **integer**)

Cross product

Forms all possible pairs of tuples from the two relations

Join operations

- Combines $\times, \sigma_c, \pi_l$ into a single op
- Simple relational algebra expressions

Inner joins

- Eliminates tuples that do not satisfy matching criteria (i.e. selection)
- Is a selection from cross product

θ-Join

$$R \bowtie_\theta S = \sigma_\theta(R \times S)$$

Equi Join

Like θ-Join, but θ must only involve $=$

Natural Join

Like equi join (i.e. only equality operator), but

- Join is performed over common attributes of R and S
- If there are no common attributes, acts like a cross product, since selection condition c is vacuously true
- Output relation keeps one copy of common attributes

Formally,
$$R \bowtie S = \pi_l(R \bowtie_c \rho_{b_i \leftarrow a_i, \cdots, b_k \leftarrow a_k}(S))$$
where

- $A = \{a_i, \cdots, a_k\}$ is the set of common attributes of R and S
- $c = (a_i = b_i) \wedge \cdots \wedge (a_k = b_k)$
- l = list of (attr. of R + attr. of S not in A)

Outer joins

- Inner join + dangling tuples
- A **dangling tuple** is a tuple that doesn't satisfy the inner join condition, i.e. foreign key not referenced in the relation.

Steps

- Perform inner join $M = R \bowtie_\theta S$
- To M, add dangling tuples from
$$\begin{cases} R & \text{in left outer join} ⟕_\theta \\ S & \text{in right outer join} ⟖_\theta \\ R \text{ and } S & \text{in full outer join} ⟗_\theta \end{cases}$$
- Pad missing attribute values with **null**

Formal definitions

- Set of dangling tuples in R, with respect to $R \bowtie_\theta S$
$$\text{dangle}(R \bowtie_\theta S) \subseteq R$$
- $null(R)$ is a n-compoennt tuple of **null** values, where n is the number of attributes in R
- Left outer join ($R ⟕_\theta S$) $= (R \bowtie_\theta S) \cup (\text{dangle}(R \bowtie_\theta S) \times \{\text{null}(S)\})$
- Right outer join ($R ⟖_\theta S$) $= (R \bowtie_\theta S) \cup (\{\text{null}(R)\} \times \text{dangle}(S \bowtie_\theta R))$
- Full outer join ($R ⟗_\theta S$)
$$= (R \bowtie_\theta S) \cup \Big((\text{dangle}(R \bowtie_\theta S) \times \{\text{null}(S)\})$$
$$\cup\, (\{\text{null}(R)\} \times \text{dangle}(S \bowtie_\theta R))\Big)$$

Natural outer joins

- Like natural inner joins
- Only equality operator used for condition
- Join is performed over common atributes of R and S
- Output relation keeps one copy of common attributes

Complex expressions

There are multiple ways to formulate a query to get the same result, e.g.

- Order of joins
- Order of selection (before/after join)
- Additional projections to minimize intermediate results

Invalid expressions

- Attribute no longer available after projection
$$\sigma_{\text{role}=\text{'dev'}}(\pi_{\text{name,age}}(Employees))$$
- Attribute no longer available after renaming
$$\sigma_{\text{role}=\text{'dev'}}(\rho_{\text{position} \leftarrow \text{role}}(Employees))$$
- Incompatible attribute types
$$\sigma_{\text{age}=\text{role}}(Employees)$$

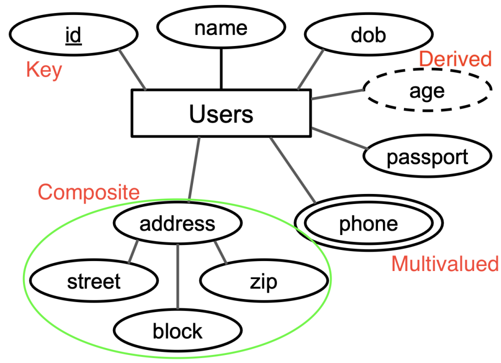
ER MODEL

Entity

- Objects that are distinguishable from other objects
- **Entity set:** Collection of entities of the same type

Attribute

- Specific information describing an entity
- **Key attr** uniquely identifies each entity
- **Composite attr** composed of multiple other attributes
- **Multivalued attr** may consist of more than one value for a given entity
- **Derived attr** derived from other attributes



Relationship

Association among two or more entities

Relationship set

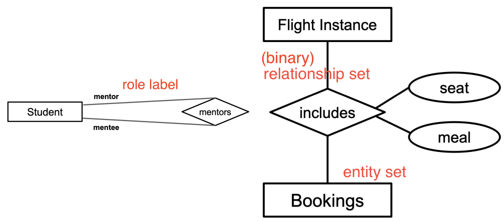
- Collection of relationships of the same type
- Can have their own attributes that further describe the relationship
- $Key(E_i)$ is the attributes of the selected key of entity set E_i

Role

- Describes an entity set's participation in a relationship
- Explicit role label only in case of ambiguities (e.g. same entity set participates in same relationship more than once)

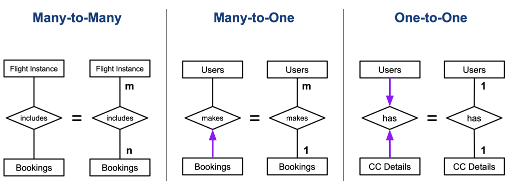
Degree

- An n-ary relationship set involves n entity roles, where n is the degree of the relationship set
- Typically binary or ternary



Cardinality constraints

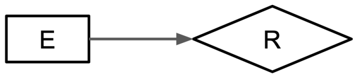
- **Upper bound** for entity's participation



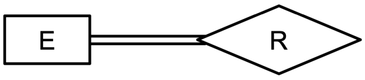
Participation constraints

- **Lower bound** for entity's participation
- Partial (default): participation not mandatory
- Total: mandatory (at least 1)

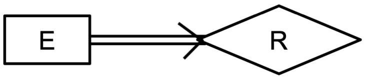
Standard



At most one

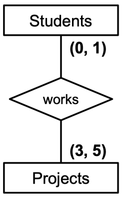


At least one



Exactly one

Alternative



A student works on exactly 1 project, or no project at all.

A project is assigned to teams comprising 3 to 5 students.

Implementation

Many-to-Many Represent relationship set with a table

Many-to-One

1. Represent relationship set between A and B with a table (A_{id}, B_{id}). Make the ID of the total participation entity set the primary key
2. Combine rel. set and total particiation entity set into one table

One-to-One

1. Represent relationship set between A and B with a table (A_{id}, B_{id}). One is unique not null, and the other is the primary key
2. Combine relationship set and either entity set into one table

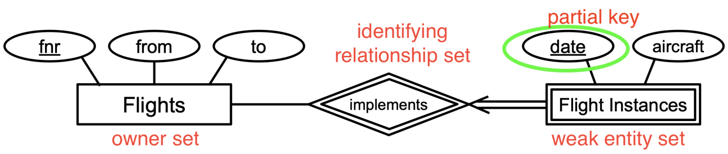
Dependency constraints

Weak entity sets

- Entity set that does not have its own key
- Can only be uniquely identified by considering primary key of owner entity
- Existence depends on existence of owner entity

Partial key

- Set of attributes of weak entity set that uniquely identifies a weak entity, for a given owner entity



Requirements

- Many-to-one relationship from weak entity set to owner entity set
- Weak entity set must have total participation in identifying relationship

Relational mapping

- Entity set → table
- Composite/multivalued attributes:
  1. Convert to single-valued attributes
  2. Additional table with FK constraint
  3. Convert to a single-valued attribute (e.g. comma separated string)

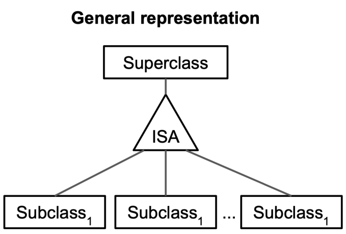
ISA Hierarchies

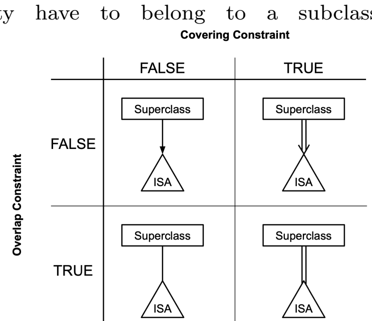
- Used to model generalization/specialization of entity sets

Constraints

Overlap Can a superclass entity belong to multiple subclasses?

Covering Does a superclass entity have to belong to a subclass?



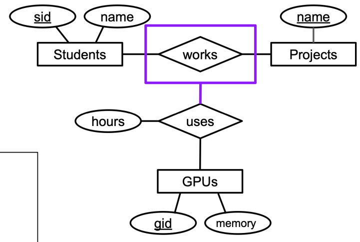


Aggregation

- Abstraction that treats relationships as higher-level entities

Schema definition of "uses"

- Primary key of aggregation relationship ➜ (sid, pname)
- Primary key of associated entity set "GPUs" ➜ gid
- Descriptive attributes of "uses" ➜ hours




```
CREATE TABLE Uses (
    gid      INTEGER,
    sid      CHAR(20),
    pname    VARCHAR(50),
    hours    NUMERIC,
    PRIMARY KEY (gid, sid, pname),
    FOREIGN KEY (gid) REFERENCES GPUs (gid),
    FOREIGN KEY (sid, pname) REFERENCES works (sid, pname)
);
```

FUNCTIONS AND PROCEDURES

```
-- Function
CREATE OR REPLACE FUNCTION <name>
(<param> <type>, ...)
RETURNS <type> AS $$
  <code>
$$ LANGUAGE <sql | plpgsql>;
-- Procedure
CREATE OR REPLACE PROCEDURE <name>
(<param> <type>, ...) AS $$
  <code>
$$ LANGUAGE <sql | plpgsql>;
```

- `CREATE OR REPLACE` helps to re-declare function/procedure if already previously defined
- Code is enclosed within `$$`
- Call a function: `SELECT * FROM swap(2, 3);`
- Call a procedure: `CALL transfer('Alice', 'Bob', 100);`

Return types

Return	Type
Single tuple from table	`<table_name>`
Set of tuples from table	`SET OF <table_name>`
Single new tuple	`RECORD`
Set of new tuples	`SET OF RECORD` or `TABLE(c VARCHAR, x INT)`
No return value	`VOID`, or use `PROCEDURE` instead of `FUNCTION`
Trigger	`TRIGGER`

Control structures

Variables

- `DECLARE [<var> <type>]` (1 or more when `DECLARE` keyword is present)
- `<var> := <expr>`

Selection

- `IF ... THEN ... [ELSIF ... THEN ...] [ELSE ...] END IF` (0 or more `ELSIF`)

Repetition

- `LOOP ... END LOOP`, and `EXIT ... WHEN ...` (conditional exit)
- `WHILE ... LOOP ... END LOOP`
- `FOR ... IN ... LOOP ... END LOOP`
- `1..10` (range, inclusive)

Block

- `BEGIN ... END`
- For plpgsql, code in the BEGIN-END block is in a transaction

Examples

Note: `INOUT` specifies that the param is both an input and output param

Function

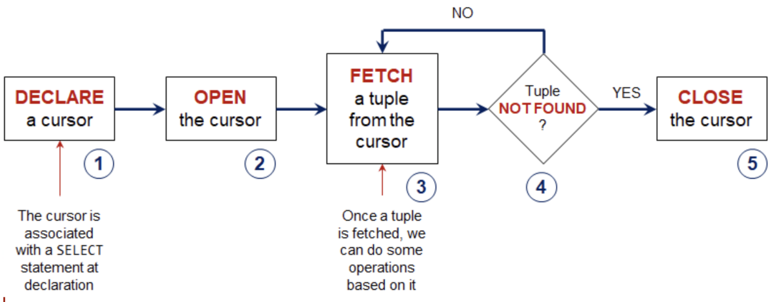
```
CREATE OR REPLACE FUNCTION
    swap(INOUT val1 INT, INOUT
    val2 INT)
RETURNS RECORD AS $$
DECLARE
  temp INT;
BEGIN
  temp := val1;
  val1 := val2;
  val2 := temp;
END;
$$ LANGUAGE plpgsql;
```

Procedure

```
CREATE OR REPLACE PROCEDURE
    transfer(
 src TEXT, dst TEXT,
 amt NUMERIC
) AS $$
  UPDATE Accounts
  SET balance = balance - amt
  WHERE name = src;
  UPDATE Accounts
  SET balance = balance + amt
  WHERE name = dst;
$$ LANGUAGE sql;
```

Cursor

- Declare, Open, Fetch, Check (repeat), Close
- `FETCH [PRIOR | FIRST | LAST | ABSOLUTE n] [FROM] <cursor> INTO <var>`



Question

Given the table "Scores" from before, write a function to perform the following task:

1. Sort the students in "Scores" in *descending* order of their `Mark` *(break ties arbitrarily)*
2. For each student, compute the *difference* between his/her `Mark` and the `Mark` of the previous student
   - If there is no previous student, use `NULL`

Solution

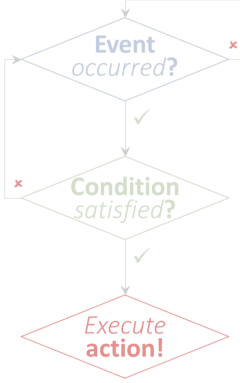
```
CREATE OR REPLACE FUNCTION score_gap()
RETURNS TABLE(name TEXT, mark INT, gap INT) AS $$
DECLARE
  curs CURSOR FOR (SELECT * FROM Scores ORDER BY Mark DESC);
  r RECORD; prev INT;
BEGIN
  prev := -1; OPEN curs;
  LOOP
    FETCH curs INTO r;
    EXIT WHEN NOT FOUND;
    name := r.Name; mark := r.Mark;
    IF prev >= 0 THEN gap := prev - mark;
    ELSE              gap := NULL;
    END IF;
    RETURN NEXT; -- insert into output
    prev := r.mark;
  END LOOP;
  CLOSE curs;
END;
$$ LANGUAGE plpgsql;
```

Explanation

1. Declare a ***cursor*** associated with a `SELECT` statement
   - `r` is a `RECORD` to store previous row
2. ***Open*** the cursor which executes the SQL statement and let the cursor point to the ***beginning*** of the result
3. ***Fetch*** a tuple from the cursor by reading the ***next*** tuple from cursor and assign into the variable
4. If the `FETCH` operation did not get any tuple, the loop ***terminates***
   - Otherwise, perform the ***main operation*** and insert into output
5. ***Close*** the cursor to release the resources allocated

TRIGGERS

- Note: cannot `CREATE OR REPLACE TRIGGER`. Need to `DROP TRIGGER`




```
-- Trigger
CREATE TRIGGER <name>
<timing> <event> ON <table>
FOR EACH <granularity>
  [ WHEN (<condition>) ]
EXECUTE FUNCTION <f_name>();
-- Trigger function
CREATE OR REPLACE FUNCTION <f_name>()
RETURNS TRIGGER AS $$
BEGIN
  <code>
END;
$$ LANGUAGE plpgsql;
```

Trigger options

Events

- `INSERT ON <table>`
- `DELETE ON <table>`
- `UPDATE [ OF <column> ] ON <table>`
- `INSERT OR DELETE OR UPDATE ON <table>`
- Alternatively, use `TG_OP` variable. Is set to `'INSERT'` | `'DELETE'` | `'UPDATE'`

Timings

- `AFTER/BEFORE` (after/before event)
- `INSTEAD OF` (replaces event, only for `VIEWS`)

Granularity

- `FOR EACH ROW` (for each tuple encountered)
- `FOR EACH STATEMENT` (for each statement)

Effect of return value

- `OLD` / `NEW`: Modified row before / after the triggering event

Events + Timings	NULL tuple	Non-NULL tuple t
`BEFORE INSERT`	No insertion	t is inserted
`BEFORE UPDATE`	No update	t is the updated tuple
`BEFORE DELETE`	No deletion	Deletion proceeds as normal
`AFTER`	No effect	No effect

Granularity

- In `FOR EACH STATEMENT`, doing `RETURN NULL` will not do anything
- Need to use `RAISE EXCEPTION` to stop the operation

Trigger condition

- Use `WHEN()` for conditional check whether a trigger should run
- e.g. `WHEN (NEW.StuName = 'Adi')`

Usage

- No `SELECT` in `WHEN()`
- No `OLD` in `WHEN()` for `INSERT`
- No `NEW` in `WHEN()` for `DELETE`
- No `WHEN()` for `INSTEAD OF`

Deferred triggers

- Triggers that are checked only at the end of a transaction
- `CONSTRAINT` + `DEFERRABLE` together indicate that trigger can be deferred
- Only works with `AFTER` and `FOR EACH ROW`
- Default is `IMMEDIATE`

```
CREATE CONSTRAINT TRIGGER <name>
AFTER <event> ON <table>
FOR EACH ROW
  [ WHEN (<condition>) ]
  [ DEFERRABLE INITIALLY [ DEFERRED | IMMEDIATE ] ]
EXECUTE FUNCTION <func_name>();
```

Multiple triggers

- Activation order for the same event on the same table:

1. `BEFORE` statement-level triggers
2. `BEFORE` row-level triggers
3. `AFTER` row-level triggers
4. `AFTER` statement-level triggers

- Within the same category, triggers are activated in alphabetical order
- If `BEFORE` row-level trigger returns `NULL`, then subsequent triggers on the same row are omitted

FUNCTIONAL DEPENDENCIES

Basic terminology

Reading FDs $X \to Y$ reads: X (functionally) determines Y | Y is functionally dependent on X | X implies Y (casual)

Instance An instance r (a table) of a relation R satisifes the FD $\sigma : X \to Y$ with $X \subset R$ and $Y \subset R$, $\iff$ if two tuples of r agree on their X-values, then they agree on their Y-values

Valid instance An instance r of relation R is a valid instance of R with $\Sigma \iff$ it satisfies Σ

Violations An instance r of relation R violates a set of FDs $\Sigma \iff$ does not satisfy Σ

Holds

- A relation R with a set of FDs Σ, R with Σ, refers to the set of valid instances of R wrt. to the FDs in Σ
- When a set of FDs Σ holds on a relation R, only consider the valid instances of R with Σ

Trivial $X \to Y$ is trivial $\iff Y \subset X$

Non-trivial $X \to Y$ is non-trivial $\iff Y \not\subset X$

Completely non-trivial $X \to Y$ is completely non-trivial $\iff Y \neq \emptyset$ and $Y \cap X = \emptyset$

Key terminology

Superkey Let $S \subset R$ be a set of attributes of R. S is a superkey of $R \iff S \to R$

Candidate key A superkey such that no proper subset is also a superkey

Primary key Chosen candidate key, or the candidate key if there is only one

Prime attribute An attribute that appears in some candidate key of R with Σ. If not, then it is a non-prime attribute

FD terminology

Closure Let Σ be a set of FDs of a relation R. The closure of Σ, denoted Σ^+, is the set of all FDs logically entailed by the FDs in Σ

Equivalence Two FDs are equivalent $\iff$ have the same closure

Cover Σ_1 is a cover of Σ_2 (and vice versa) $\iff$ their closure are equivalent

Closure of a set of attributes Let Σ be a set of FDs of a relation R. The closure of a set of attributes $S \subset R$, denoted S^+, is the set of all attributes that are functionally dependent on S (i.e. what S implies)

$$S^+ = \{A \in R \mid \exists(S \to \{A\}) \in \Sigma^+\}$$

Computing attribute closures

- Check if any attribute doesn't appear in the RHS of any FD. These attributes must appear in the key
- Compute attribute closure starting with singular attributes. Then compute for 2 elements, 3 elements and so on.
- Note all candidate keys in the process
- If current set of attributes is a superset of some previously seen, candidate key, can skip

Armstrong axioms

Reflexivity $\forall X, Y \subset R \left((Y \subset X) \Rightarrow (X \to Y) \right)$

Augmentation
$\forall X, Y, Z \subset R \left((X \to Y) \Rightarrow (X \cup Z \to Y \cup Z) \right)$

Transitivity
$\forall X, Y, Z \subset R \left((X \to Y) \wedge (Y \to Z) \Rightarrow (X \to Z) \right)$

Remarks

- **Sound**: The rule only generates elements of Σ^+ when applied to Σ
- **Complete**: The rule(s) generate(s) all elements of Σ^+ when applied to Σ
- The three inference rules are (individually) sound
- The Armstrong axioms are (together) sound and complete

Additional rules

Must be derived during exam

Weak augmentation

If $X \to Y$, then $X \cup Z \to Y$

Proof

1. $X \to Y$ (given)
2. We know that $X \subset X \cup Z$
3. $X \cup Z \to X$ (reflexivity)
4. $X \cup Z \to Y$ (trans. of 3 and 1)

Union

If $X \to Y$ and $X \to Z$, then $X \to Y \cup Z$

Proof

1. $X \to Y$ (given)
2. $X \to Z$ (given)
3. $X \to X \cup Z$ (aug. 2 and X)
4. $X \cup Z \to Y \cup Z$ (aug. 1 and Z)
5. $X \to Y \cup Z$ (trans. of 3 and 4)

Decomposition

If $X \to Y \cup Z$, then $X \to Y$ and $X \to Z$

Proof

1. $X \to Y \cup Z$ (given)
2. $Y \cup Z \to Y$ (reflexivity)
3. $X \to Y$ (trans. of 1 and 2)

Composition

If $X \to Y$ and $A \to B$, then $X \cup A \to Y \cup B$

Proof

1. $X \to Y$ (given)
2. $A \to B$ (given)
3. $X \cup A \to Y \cup A$ (aug. 1 and A)
4. $X \cup A \to Y$ (decomp. of 3)
5. $X \cup A \to X \cup B$ (aug. 2 and X)
6. $X \cup A \to B$ (decomp. of 5)
7. $X \cup A \to Y \cup B$ (union 4 and 6)

Pseudo-transitivity

If $X \to Y$ and $Y \cup Z \to W$, then $X \cup Z \to W$

Proof

1. $X \to Y$ (given)
2. $Y \cup Z \to W$ (given)
3. $X \cup Z \to Y \cup Z$ (aug. of 1 and Z)
4. $X \cup Z \to W$ (trans. of 3 and 2)

Minimal cover

Definition

A set Σ of FDs is minimal if and only if

- RHS of each FD in Σ is minimal, i.e. each FD is of the form $X \to \{A\}$
- LHS of each FD in Σ is minimal, i.e. for every FD in Σ of the form $X \to \{A\}$, there is no FD $Y \to \{A\}$ such that $Y \subset X$
- The set is minimal, i.e. no FD in Σ can be derived from other FDs in Σ

Misc

- A minimal cover of a set of FDs Σ is a set of FDs Σ' that is both minimal and equivalent to Σ
- Every set of FDs has a minimal cover

Algorithm

1. Simplify RHS of every FD (by splitting FDs so that RHS of each FD is a singleton)
2. Simplify LHS of every FD (for each FD, if a subset of LHS can imply RHS, then replace LHS with the subset)
3. Remove redundant FDs (for each FD, start from LHS. if this FD can be derived using only other FDs, then remove it)
4. (If compact cover is desired) Combine FDs with same LHS

Reachability

- The algorithm always finds a minimal cover
- Some minimal covers may be unreachable
- To reach all minimal covers, the algorithm needs to start from Σ^+

Anomalies

userid	domain	department	faculty	language
tanh	comp.sut.edu	computer science	computing	JavaScript
tanh	comp.sut.edu	computer science	computing	Python
ami	med.sut.edu	pharmacy	medicine	R

- Department $\to$ faculty is a FD in this example
- **Redundant storage**: The faculty of a department is repeated for every student of the department, and every time the student is proficient in a language
- **Update anomaly**: When 2 rows of the table have the same value for the column `department` but different values for the column `faculty`, violating the FD
- **Deletion anomaly**: If we delete the last row, we may forget that we have a department of pharmacy, and a faculty of medicine
- **Insertion anomaly**: We cannot record that the department of social science exists and the faculty of liberal arts exists, because there is no student from this department or this faculty

Solution

- In all cases, the solution is to remove faculty from the original table, and create a new table with department and faculty
- In the case of the update anomaly, to enforce the FD, we also need to make department the primary key of the new table

Normalization

Normal forms

- Recognize designs that enforce FDs through main SQL constraints (PK, unique, not null, FK)
- Protect data against anomalies

Normalization

Transform (decompose) a poor design into one that enforces FDs by means of the main SQL constraints

Boyce-Codd Normal Form

A relation R with a set of FDs Σ is in BCNF $\iff$ for every FD $X \to \{A\} \in \Sigma^+$,

- either $X \to \{A\}$ is trivial, or
- X is a superkey

Check if decomp. set of relations is in BCNF

1. Compute attribute closures of the original set, and project FDs to each relation R_i
2. If some R_i is not in BCNF, then FALSE. Else TRUE

Decomposition

Terminology

Decomposition A decomposition of table R is a set of tables $\{R_1, R_2, \cdots, R_n\}$ such that $R = R_1 \cup \cdots \cup R_n$

Binary decomposition Decomp with $n = 2$

Lossless-join definitions

- A binary decomp is lossless-join $\iff$ full outer natural join of its two fragments equal the initial table. Otherwise it is lossy
- A binary decomp of R into R_1 and R_2 is lossless-join if $R = R_1 \cup R_2$ and either $R_1 \cap R_2 \to R_1$ or $R_1 \cap R_2 \to R_2$
- A decomp is lossless-join if there exists a sequence of binary lossless-join decomp that generates that decomp

Projected FDs A set Σ of projected FDs on R', from R with Σ where $R' \subset R$, is the set of FDs equivalent to the set of FDs $X \to Y$ in Σ^+ such that $X \subset R'$ and $Y \subset R'$

Dependency preserving A decomp of R with Σ into $R_1, R_2, \cdots, R_n$ with respective projected FDs $\Sigma_1, \Sigma_2, \cdots, \Sigma_n$ is dependency preserving $\iff$ $\Sigma^+ = (\Sigma_1 \cup \cdots \cup \Sigma_n)^+$

Check lossless-join

1. Compute attribute closures of the original set
2. For some pair $(R_i, R_j), i \neq j$,
   - Check that $R_i \cap R_j \to R_i$ or $R_i \cap R_j \to R_j$
   - If yes, then replace R_i, R_j with $R_i \cup R_j$ and repeat Step 2-3.
   - If no, then try next pair
3. If there was a sequence of unions that resulted in a single relation that has all attributes, then TRUE. Else FALSE

Check dependency-preserving

1. Let R be the original set of relations. Can replace R with minimal cover if found. Let Y be the union of projected FDs to each relation
2. For each FD in R, check that we can derive it in Y. If some FD could not be derived, then that FD was not preserved, then FALSE. Else TRUE

Decomposition algorithm

- Guarantees lossless decomp, but may not be dependency preserving
- Can get different results depending on order of FD chosen

Let $X \to Y$ be a FD in Σ that violates the BCNF definition (not trivial, and X not superkey). Use it to decompose R into R_1 and R_2:

- $R_1 = X^+$
- $R_2 = (R - X^+) \cup X$

Then, check whether R_1 and R_2 with respective projected FDs Σ_1 and Σ_2 are in BCNF. Repeat decomp algo for the fragments which are not.

3NF Definition

A relation R with a set of FDs Σ is in 3NF $\iff$ for every FD $X \to \{A\} \in \Sigma^+$,

- $X \to \{A\}$ is trivial, or
- X is a superkey, or
- A is a prime attribute

Note that BCNF implies 3NF

3NF Synthesis

Guarantees a lossless, dependency preserving decomp in 3NF

- For each FD $X \to Y$ in the compact minimal cover, create a relation $R_i = X \cup Y$ unless it already exists, or is subsumed by another relation
- If none of the created relations contains one of the keys, pick a candidate key and create a relation with that candidate key