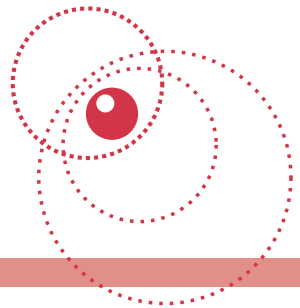

CS4221: Database Applications Design and Tuning

Week 4: The Chase

4 February 2016

Pierre Senellart

TELECOM ParisTech

Chasing IDs and MVDs

Let R be a relation schema and Σ a set of functional and multi-valued dependencies on R.

The Chase is an algorithm that solves the decision problem of whether a functional or multi-valued dependency σ is implied by Σ in R:

$$\Sigma \overset{?}{\models} \sigma$$

In other words:

$$\forall r \text{ instance of } R \quad r \models \Sigma \quad \overset{?}{\Longrightarrow} \quad r \models \sigma$$

1. $\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$

1. $\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$
2. $\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$

1. $\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$
2. $\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$
3. $\{\{A\} \twoheadrightarrow \{B, C\}, \{C, D\} \rightarrow \{B\}\} \stackrel{?}{\models} \{A\} \rightarrow \{B\}$

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$$

Create an instance r on the schema $\{A, B, C, D\}$ with two tuples and distinct values for all attributes.

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$$

We want to chase $\{A\} \to \{C\}$.

Make the A-values the same.

$$a_1 = a_2$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$$

Use $\{A\} \twoheadrightarrow \{B, C\}$. Create two new tuples by copying the two tuples that have the same A-value but swapping their values for $\{B, C\}$. The multi-valued dependency generates tuples. It is a tuple-generating dependency.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1
a_1	b_1	c_1	d_2

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \rightarrow \{C\}\} \stackrel{?}{\models} \{A\} \rightarrow \{C\}$$

Use $\{D\} \rightarrow \{C\}$. For each pair of tuple with the same D-value, make their C-value the same: $c_1 = c_2$ The functional dependency generates values. It is an equality-generating dependency.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_1	d_2

$$\{\{A\} \twoheadrightarrow \{B, C\}, \{D\} \to \{C\}\} \stackrel{?}{\models} \{A\} \to \{C\}$$

There is nothing else to do. We observe that r satisfies $\{A\} \to \{C\}$.

$$r \models \{A\} \to \{C\}$$

Therefore the answer is yes.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_1	d_2

r also satisfies $\{D\} \rightarrow \{A\}$ but this is a coincidence. We can only answer the question about $\{A\} \rightarrow \{C\}$.

Another chase would be needed for $\{D\} \rightarrow \{A\}$.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_1	d_2

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \overset{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \overset{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

We want to chase $\{A\} \twoheadrightarrow \{C\}$.
Make the A-values the same.

$$a_1 = a_2$$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \overset{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

Use $\{A\} \twoheadrightarrow \{B\}$.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_2	d_2

$$R = \{A, B, C, D\}$$

$$\{\{A\} \twoheadrightarrow \{B\}, \{B\} \twoheadrightarrow \{C\}\} \stackrel{?}{\models} \{A\} \twoheadrightarrow \{C\}$$

Use $\{B\} \twoheadrightarrow \{C\}$ (twice).

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_2	d_1
a_1	b_1	c_1	d_2
a_1	b_2	c_1	d_2
a_1	b_2	c_2	d_1

There is nothing else to do.

$$r \models \{A\} \twoheadrightarrow \{C\}$$

Therefore the answer is yes

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_2	d_1
a_1	b_1	c_1	d_2
a_1	b_2	c_1	d_2
a_1	b_2	c_2	d_1

$$\{\{A\} \twoheadrightarrow \{B,C\}, \{C,D\} \rightarrow \{B\}\} \stackrel{?}{\models} \{A\} \rightarrow \{B\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2

$$\{\{A\} \twoheadrightarrow \{B,C\}, \{C,D\} \rightarrow \{B\}\} \stackrel{?}{\models} \{A\} \rightarrow \{B\}$$

Use $\{A\} \twoheadrightarrow \{B,C\}$.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1
a_1	b_1	c_1	d_2

There is nothing else to do.

$$r \not\models \{A\} \rightarrow \{B\}$$

Therefore the answer is No

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1
a_1	b_1	c_1	d_2

We have built a counter-example.

The chase is a very powerful tool: it can be used to prove that a functional or multi-valued dependency is satisfied!

The chase is a very powerful tool: it can be used to prove that a functional or multi-valued dependency is satisfied!

Theorem ([Maier et al., 1979])

The chase of FDs and MVDs terminates after a finite number of operations, and always builds a counter example if and only if such an example exists.

The chase is a very powerful tool: it can be used to prove that a functional or multi-valued dependency is satisfied!

Theorem ([Maier et al., 1979])

The chase of FDs and MVDs terminates after a finite number of operations, and always builds a counter example if and only if such an example exists.

Why? We will see this is an instance of a much more general result.

Let Σ be a set of functional and multi-valued dependencies on a relation schema R. Let σ be a functional or multi-valued dependency.

$$\sigma = X \rightarrow Y \text{ or } \sigma = X \twoheadrightarrow Y$$

Let Σ be a set of functional and multi-valued dependencies on a relation schema R. Let σ be a functional or multi-valued dependency.

$$\sigma = X \rightarrow Y \text{ or } \sigma = X \twoheadrightarrow Y$$

1. Create a table r with schema R with two tuples with all different values.

Let Σ be a set of functional and multi-valued dependencies on a relation schema R. Let σ be a functional or multi-valued dependency.

$$\sigma = X \to Y \text{ or } \sigma = X \twoheadrightarrow Y$$

1. Create a table r with schema R with two tuples with all different values.
2. For each $A \in X$, make the A-values the same (choosing new and different values for each A, though).

1. For each functional dependency $Z \rightarrow V \in \Sigma$:
   - If there are tuples in the table with same Z-value, then set their V-values to be the same.
2. For each multi-valued dependency $Z \twoheadrightarrow V \in \Sigma$:
   - If there are two tuples in the table with same Z-value, then add two new tuples with all the same values except for their V-values that are swapped.

On termination (which always happens):

$$r \models \sigma \quad \text{is equivalent to} \quad \Sigma \models \sigma$$

You simply need to check whether or not r satisfies the functional or multi-valued dependency σ that you were chasing.

Recall that on a relation schema $R = X \cup Y \cup Z$ with X, Y, Z distinct, a MVD $X \twoheadrightarrow Y$ holds in r iff:

$$\pi_{X \cup Y}(r) \bowtie \pi_{X \cup Z}(r) = r$$

This means MVDs can be used to test if a relation schema $R = X \cup Y \cup Z$ (satisfying a set of dependencies Σ) can be decomposed into two relations $R_1 = X \cup Y$ and $R_2 = X \cup Z$ in a lossless manner:

Recall that on a relation schema $R = X \cup Y \cup Z$ with X, Y, Z distinct, a MVD $X \twoheadrightarrow Y$ holds in r iff:

$$\pi_{X \cup Y}(r) \bowtie \pi_{X \cup Z}(r) = r$$

This means MVDs can be used to test if a relation schema $R = X \cup Y \cup Z$ (satisfying a set of dependencies Σ) can be decomposed into two relations $R_1 = X \cup Y$ and $R_2 = X \cup Z$ in a lossless manner:

Use the chase to decide whether $\Sigma \stackrel{?}{\models} X \twoheadrightarrow Y$

(or, equivalently, $\Sigma \stackrel{?}{\models} X \twoheadrightarrow Z$)

- Logical formalism to represent queries
- Can be also used to represent dependencies (FDs, MVDs, and beyond)
- Basically: First-order logic with relation symbols
- Codd's theorem: the relational calculus has the same expressive power as the relational algebra
- SQL is actually closer to the relational calculus than to the relational algebra
- Two variants: tuple calculus and domain calculus; we focus on domain calculus

Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a relational schema, i.e., a set of relation schemas. Each relation schema R_i is a sequence of attributes $(A_{i1}, \ldots, A_{ik})$ (not just a set).

A query is of the form $Q(x_1, \ldots, x_n) = \phi(x_1, \ldots, x_n)$ where ϕ is an arbitrary first-order formula with free variables $x_1 \ldots x_n$, using the following constructs:

- $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$: Boolean connectors
- $\exists x$, $\forall y$: quantifiers
- $R_i(z_1, \ldots, z_k)$: fact, to be read "the tuple $(z_1, \ldots, z_k)$ is in the relation R_i"
- $z_p = z_k$: equality between two variables

$S = \{\text{Course}, \text{Student}, \text{Registered}\}$

$\text{Course} = (\text{cid}, \text{title}),\ \text{Student} = (\text{sid}, \text{fname}, \text{lname}),$

$\text{Registered} = (\text{rcid}, \text{rsid}, \text{year}),$

$S = \{\text{Course}, \text{Student}, \text{Registered}\}$

$\text{Course} = (\text{cid}, \text{title}),\ \text{Student} = (\text{sid}, \text{fname}, \text{lname}),$
$\text{Registered} = (\text{rcid}, \text{rsid}, \text{year}),$

```
SELECT fname, lname
FROM Course, Student, Registered
WHERE cid=rcid AND sid=rsid
  AND year=2016 AND title='Databases'
```


$S = \{\text{Course}, \text{Student}, \text{Registered}\}$

$\text{Course} = (\text{cid}, \text{title}),\ \text{Student} = (\text{sid}, \text{fname}, \text{lname}),$

$\text{Registered} = (\text{rcid}, \text{rsid}, \text{year}),$

```
SELECT fname, lname
FROM Course, Student, Registered
WHERE cid=rcid AND sid=rsid
  AND year=2016 AND title='Databases'
```

$$\pi_{\text{fname,lname}}(\sigma_{\text{title}=\text{`Databases'}}(\text{Course}) \bowtie_{\text{cid}=\text{rcid}} \sigma_{\text{year}=2016}(\text{Registered}) \bowtie_{\text{rsid}=\text{sid}} \text{Student})$$

$S = \{\text{Course}, \text{Student}, \text{Registered}\}$

$\text{Course} = (\text{cid}, \text{title}), \text{Student} = (\text{sid}, \text{fname}, \text{lname}),$
$\text{Registered} = (\text{rcid}, \text{rsid}, \text{year}),$

```
SELECT fname, lname
FROM Course, Student, Registered
WHERE cid=rcid AND sid=rsid
  AND year=2016 AND title='Databases'
```

$$\pi_{\text{fname,lname}}(\sigma_{\text{title}=\text{`Databases'}}(\text{Course}) \bowtie_{\text{cid}=\text{rcid}} \sigma_{\text{year}=2016}(\text{Registered}) \bowtie_{\text{rsid}=\text{sid}} \text{Student})$$

$$Q(f,l) = \exists c, s\ \text{Course}(c, \text{`Databases'}) \wedge \text{Student}(s,f,l) \wedge \text{Registered}(c,s,2016)$$

$R = (A_1, \ldots A_n)$; $\{A_1\} \rightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$

$$R(a_1, a_2, a_3, a_4, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, a'_4, \ldots, a'_n) \Longrightarrow a_2 = a'_2 \wedge a_3 = a'_3$$

$R = (A_1, \ldots A_n)$; $\{A_1\} \rightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$

$$R(a_1, a_2, a_3, a_4, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, a'_4, \ldots, a'_n) \Longrightarrow a_2 = a'_2 \wedge a_3 = a'_3$$

Such a formula is an instance of a more general formula of the form

$$\forall x_1 \ldots \forall x_m \quad \phi(x_1 \ldots x_m) \Rightarrow \psi(x_1 \ldots x_m)$$

where $\phi(x_1 \ldots x_m)$ is a conjunction of facts using variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m)$ is a conjunction of equalities using variables $x_1 \ldots x_m$.

$R = (A_1, \ldots A_n)$; $\{A_1\} \rightarrow \{A_2 A_3\}$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$$

$$R(a_1, a_2, a_3, a_4, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, a'_4, \ldots, a'_n) \Longrightarrow a_2 = a'_2 \wedge a_3 = a'_3$$

Such a formula is an instance of a more general formula of the form

$$\forall x_1 \ldots \forall x_m \quad \phi(x_1 \ldots x_m) \Rightarrow \psi(x_1 \ldots x_m)$$

where $\phi(x_1 \ldots x_m)$ is a conjunction of facts using variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m)$ is a conjunction of equalities using variables $x_1 \ldots x_m$.
Such formulas are called equality-generating dependencies (EGDs).
FDs have the particularity of having:

- only 2 facts on the left-hand side
- only 1 relation mentioned

$R = (A_1, \ldots A_n)$; $\{A_1\} \twoheadrightarrow \{A_2 A_3\}$ can be expressed as:

$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$

$$R(a_1, a_2, a_3, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, \ldots, a'_n) \Longrightarrow R(a_1, a'_2, a'_3, a_4, \ldots, a_n)$$

$R = (A_1, \ldots A_n)$; $\{A_1\} \twoheadrightarrow \{A_2 A_3\}$ can be expressed as:

$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$

$$R(a_1, a_2, a_3, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, \ldots, a'_n) \Longrightarrow R(a_1, a'_2, a'_3, a_4, \ldots, a_n)$$

Such a formula is an instance of a more general formula of the form

$$\forall x_1 \ldots \forall x_m \quad \phi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \, \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

where $\phi(x_1 \ldots x_m)$ is a conjunction of facts using variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m, y_1, \ldots, y_p)$ is a conjunction of facts using variables $x_1 \ldots x_m$ and $y_1 \ldots y_p$.

$R = (A_1, \ldots A_n)$; $\{A_1\} \twoheadrightarrow \{A_2 A_3\}$ can be expressed as:

$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n \forall a'_2 \forall a'_3 \ldots \forall a'_n$

$$R(a_1, a_2, a_3, \ldots, a_n) \wedge R(a_1, a'_2, a'_3, \ldots, a'_n) \Longrightarrow R(a_1, a'_2, a'_3, a_4, \ldots, a_n)$$

Such a formula is an instance of a more general formula of the form

$$\forall x_1 \ldots \forall x_m \quad \phi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \, \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

where $\phi(x_1 \ldots x_m)$ is a conjunction of facts using variables $x_1 \ldots x_m$ and $\psi(x_1 \ldots x_m, y_1, \ldots, y_p)$ is a conjunction of facts using variables $x_1 \ldots x_m$ and $y_1 \ldots y_p$.

Such formulas are called tuple-generating dependencies (TGDs).

MVDs have the particularity of having:

- only 2 facts on the left-hand side (more than 2: JDs)
- only 1 facts on the right-hand side, no $\exists$
- only 1 relation mentioned
-

FDs and MVDs are only able to express constraints (or dependencies) local to a relation

- FDs and MVDs are only able to express constraints (or dependencies) local to a relation
- In practice, one wants to express dependencies across relations, in particular foreign key constraints

- FDs and MVDs are only able to express constraints (or dependencies) local to a relation
- In practice, one wants to express dependencies across relations, in particular foreign key constraints
- The tuples of values of some attributes are included in the values of some other attributes (possibly in some other table)

- FDs and MVDs are only able to express constraints (or dependencies) local to a relation
- In practice, one wants to express dependencies across relations, in particular foreign key constraints
- The tuples of values of some attributes are included in the values of some other attributes (possibly in some other table)
- In SQL:

```
CREATE TABLE R1(
  A INT, B VARCHAR(42), ...,
  FOREIGN KEY (A,B) REFERENCES R2(C,D) )
```

- FDs and MVDs are only able to express constraints (or dependencies) local to a relation
- In practice, one wants to express dependencies across relations, in particular foreign key constraints
- The tuples of values of some attributes are included in the values of some other attributes (possibly in some other table)
- In SQL:

```
CREATE TABLE R1(
  A INT, B VARCHAR(42), ...,
  FOREIGN KEY (A,B) REFERENCES R2(C,D) )
```

- Notation: $R_1[A, B] \subseteq R_2[C, D]$

$R = (A_1, \ldots A_n)$, $S = (B_1, \ldots B_m)$; $R[A_1] \subseteq S[B_2]$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n$$

$$R(a_1, a_2, a_3, \ldots, a_n) \Longrightarrow \exists b_1 \exists b_3 \ldots \exists b_m \; S(b_1, a_1, b_3, \ldots, b_m)$$

$R = (A_1, \ldots A_n)$, $S = (B_1, \ldots B_m)$; $R[A_1] \subseteq S[B_2]$ can be expressed as:

$$\forall a_1 \forall a_2 \forall a_3 \ldots \forall a_n$$
$$R(a_1, a_2, a_3, \ldots, a_n) \Longrightarrow \exists b_1 \exists b_3 \ldots \exists b_m \; S(b_1, a_1, b_3, \ldots, b_m)$$

Such a formula is an instance of a tuple-generating dependency:

$$\forall x_1 \ldots \forall x_m \quad \phi(x_1 \ldots x_m) \Rightarrow \exists y_1 \ldots \exists y_p \; \psi(x_1 \ldots x_m, y_1, \ldots, y_p)$$

IDs have the particularity of having:

- only 1 fact on the left-hand side
- only 1 fact on the right-hand side

- The chase is not limited to FDs and MVDs
- In fact, it can be defined for an arbitrary set of TGDs and EGDs, which goes far beyond FDs, MVDs, JDs, and even IDs
- TGDs (+ EGDs) are a fundamental object of study in their own right, beyond the setting of database design. Known in other communities as *existential rules* [Baget et al., 2011], *Datalog*$^{\pm}$ [Calì et al., 2010]
- The chase is a major practical and theoretical tool to work with these objects

Let Σ be a set of TGDs and EGDs on a relational schema S. Let σ be a TGD or EGD.

Let Σ be a set of TGDs and EGDs on a relational schema S. Let σ be a TGD or EGD.

1. Create a database d with schema S with tuples the tuples appearing on the left-hand side of σ.
2. Make sure shared variables across tuples also have the same value in the initial database.

Repeat the following until you reach a fixed point (nothing changes):

1. For each EGD:
   - If there is a match of the left-hand side on the database, impose the equalities from the right-hand side on the values of the database.
2. For each TGD:
   - If there is a match of the left-hand side on the database but not of the right-hand side, add to the database the tuples indicated by the right-hand side. For existentially quantified variables, introduce new values (always fresh, never used before).

On termination (which is not guaranteed to happen):

$$r \models \sigma \quad \text{is equivalent to} \quad \Sigma \models \sigma$$

Theorem

If there are no existentially quantified variables in any dependency, the chase terminates

Theorem

If there are no existentially quantified variables in any dependency, the chase terminates

Proof.

Since no new values are introduced, there is a bounded number of tuples that can be produced by the chase! □

Theorem

If there are no existentially quantified variables in any dependency, the chase terminates

Proof.

Since no new values are introduced, there is a bounded number of tuples that can be produced by the chase! □

Can be generalized to other settings: when the TGDs do not introduce cycles, or when they introduce only "nice" cycles (e.g., weak acyclicity [Fagin et al., 2005]). But fails if we have arbitrary FDs and IDs.

Theorem

If there are no existentially quantified variables in any dependency, the chase terminates

Proof.

Since no new values are introduced, there is a bounded number of tuples that can be produced by the chase! □

Can be generalized to other settings: when the TGDs do not introduce cycles, or when they introduce only "nice" cycles (e.g., weak acyclicity [Fagin et al., 2005]). But fails if we have arbitrary FDs and IDs. Indeed, implication of a dependency by a collection of FDs and IDs is undecidable. [Chandra et al., 1981]

Theorem ([Fagin et al., 2005])

The result of the chase is a universal model: all databases that satisfy the dependencies have a subinstance that can be mapped to the chase. Said differently, a conjunction of facts is true on the chase if and only if it is true on all databases that satisfy the dependencies.

Theorem ([Fagin et al., 2005])

The result of the chase is a universal model: all databases that satisfy the dependencies have a subinstance that can be mapped to the chase. Said differently, a conjunction of facts is true on the chase if and only if it is true on all databases that satisfy the dependencies.

- Note that the result of the chase is not unique (depends on order of application), but works for any result
- When the chase does not terminate, it can still be useful: e.g., in some circumstances, it is enough to consider a bounded portion of the chase

- Answer queries on all possible completions of a database under a set of TGDs and EGDs [Calì et al., 2003, 2012]
- Data integration [Lenzerini, 2002]: TGDs and EGDs express the relationships between two schemas, and one performs query answering on one dataset based on the dependencies and a dataset in the other schema
- Data exchange [Fagin et al., 2005, Onet, 2013]: TGDs and EGDs express the relationships between two schemas, and one uses the chase to generate the translation of a dataset in the other schema

- The original paper about the chase [Maier et al., 1979]
- Chapters 8 to 11 of [Abiteboul et al., 1995], textbook reference
- The general chase, a modern view: [Fagin et al., 2005]

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0. URL http://www-cse.ucsd.edu/users/vianu/book.html.

Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011. doi: 10.1016/j.artint.2011.03.002. URL http://dx.doi.org/10.1016/j.artint.2011.03.002.

Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In Frank Neven, Catriel Beeri, and Tova Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 260–271. ACM, 2003. ISBN 1-58113-670-6. doi:

10.1145/773153.773179. URL http://doi.acm.org/10.1145/773153.773179.

Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris. Datalog+/-: A family of languages for ontology querying. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Jon Sellers, editors, *Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers*, volume 6702 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2010. ISBN 978-3-642-24205-2. doi: 10.1007/978-3-642-24206-9_20. URL http://dx.doi.org/10.1007/978-3-642-24206-9_20.

Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. doi: 10.1016/j.artint.2012.08.002. URL http://dx.doi.org/10.1016/j.artint.2012.08.002.

Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. Embedded implicational dependencies and their inference problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 342–354. ACM, 1981. doi: 10.1145/800076.802488. URL http://doi.acm.org/10.1145/800076.802488.

Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005. doi: 10.1016/j.tcs.2004.10.033. URL http://dx.doi.org/10.1016/j.tcs.2004.10.033.

Maurizio Lenzerini. Data integration: A theoretical perspective. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246. ACM, 2002. ISBN 1-58113-507-6. doi: 10.1145/543613.543644. URL http://doi.acm.org/10.1145/543613.543644.

David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4 (4):455–469, 1979. doi: 10.1145/320107.320115. URL http://doi.acm.org/10.1145/320107.320115.

Adrian Onet. The chase procedure and its applications in data exchange. In Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt, editors, *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 1–37. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. ISBN 978-3-939897-61-3. doi: 10.4230/DFU.Vol5.10452.1. URL `http://dx.doi.org/10.4230/DFU.Vol5.10452.1`.