Relational Databases

A Review

Guillaume Raschia — Université de Nantes

Last update: April 8, 2013

[Source : J. Cheney, Univ. of Edinburgh]

[Source : L. Libkin, Univ. of Edinburgh]

Databases

What is a database?

- Any collection?
- A file system?
- A collection of relational tables?
- A bunch of XML documents?
- A multimedia digital library containing text, images, music, and sound recordings?

What differentiates a big pile of data from a database?

Databases

A database is. . .

- . . . a collection of **structured data** (files, records, trees, key-values)
- . . . that provides a **high-level interface** (query/update languages)
- . . . and isolates **users** from low-level (and transient) implementation details (storage layout, query optimization, data structure traversals)

In other words, databases are **ADTs for managing large amounts of information**

Database theory

Is it all just about making SQL Server 0.3% faster ? No.

- Database technology is very important commercially, hence an alphabet soup of acronyms, buzzwords, standards, and fads
- But the underlying theory of databases is elegant and technology-independent
- In fact, database systems (in their current form) would not exist without theory

When Dinosaurs Ruled the Earth

- 1960s: Databases provided “network” (graph) or “hierarchical” (tree) data model. CODASYL, COBOL. No underlying theory or declarative semantics

When Dinosaurs Ruled the Earth

- 1960s: Databases provided “network” (graph) or “hierarchical” (tree) data model. CODASYL, COBOL. No underlying theory or declarative semantics
- 1970s: Codd proposes relational model: everything is a table, separate interface from implementation, etc.

When Dinosaurs Ruled the Earth

- 1960s: Databases provided “network” (graph) or “hierarchical” (tree) data model. CODASYL, COBOL. No underlying theory or declarative semantics
- 1970s: Codd proposes relational model: everything is a table, separate interface from implementation, etc.
- 1980s: Commercial success of RDBMS technology (Oracle, IBM, etc.)

When Dinosaurs Ruled the Earth

- 1960s: Databases provided “network” (graph) or “hierarchical” (tree) data model. CODASYL, COBOL. No underlying theory or declarative semantics
- 1970s: Codd proposes relational model: everything is a table, separate interface from implementation, etc.
- 1980s: Commercial success of RDBMS technology (Oracle, IBM, etc.)
- 1990s: Object oriented DB research popular, but fails to make significant commercial impact

When Dinosaurs Ruled the Earth

- 1960s: Databases provided "network" (graph) or "hierarchical" (tree) data model. CODASYL, COBOL. No underlying theory or declarative semantics
- 1970s: Codd proposes relational model: everything is a table, separate interface from implementation, etc.
- 1980s: Commercial success of RDBMS technology (Oracle, IBM, etc.)
- 1990s: Object oriented DB research popular, but fails to make significant commercial impact
- 2000s: XML/semistructured DB research popular, but. . . ?

When Dinosaurs Ruled the Earth

- 1960s: Databases provided “network” (graph) or “hierarchical” (tree) data model. CODASYL, COBOL. No underlying theory or declarative semantics
- 1970s: Codd proposes relational model: everything is a table, separate interface from implementation, etc.
- 1980s: Commercial success of RDBMS technology (Oracle, IBM, etc.)
- 1990s: Object oriented DB research popular, but fails to make significant commercial impact
- 2000s: XML/semistructured DB research popular, but. . . ?
- 2010s: NoSQL DB systems: cloud for scaling-up, simple makes it efficient, anti-relational

When Dinosaurs Ruled the Earth

- 1960s: Databases provided “network” (graph) or “hierarchical” (tree) data model. CODASYL, COBOL. No underlying theory or declarative semantics
- 1970s: Codd proposes relational model: everything is a table, separate interface from implementation, etc.
- 1980s: Commercial success of RDBMS technology (Oracle, IBM, etc.)
- 1990s: Object oriented DB research popular, but fails to make significant commercial impact
- 2000s: XML/semistructured DB research popular, but. . . ?
- 2010s: NoSQL DB systems: cloud for scaling-up, simple makes it efficient, anti-relational
- The future: ACID/SQL revolution with NewSQL?

Preaching to the choir

- In retrospect, widely agreed that simple, clear, compelling data model is what made RDBMSs so successful
- Lack of same is what killed network, hierarchical, OO databases
- Some feeling that XML DB work has same problem

So: Theory is—at least—as important than practice

Source material

- For relational database theory:
  - Foundations of Databases, S. Abiteboul, R. Hull, and V. Vianu, Addison-Wesley 1995.
  - Elements of Finite Model Theory, L. Libkin, Springer-Verlag 2004.
- For database research foundations:
  - Readings in Database Systems (*The Red Book*), J. H. Hellerstein, and M. Stonebraker, 4th Edition, MIT Press 2005.
- For database systems:
  - Database Systems: The Complete Book, H. Garcia-Molina, J. D. Ullman, and J. Widom, 2nd Edition, Prentice Hall 2008.
  - Database Management Systems, R. Ramakrishnan, and J. Gehrke, 3rd Edition, McGraw-Hill Science, 2002.

Overview

It is all about...

The relational data model

- Data is stored in *n*-ary relations (i.e., tables)
- Data is described using "schema" (relations are typed)
- Data is manipulated using set-at-a-time relational operators (SQL)

Behind the scenes

Key theoretical issues

- Query language semantics and logical interpretation
- Descriptive power of query language features: conjunction, disjunction, negation, quantification
- Decision problems: query equivalence, constraint entailment
- Related area: finite model theory

The relational model

Some terminology

- **Domain**: countably infinite set D of individual data values (by convention, $D = \{a, b, c, d, \ldots\}$
- Rows, records or **tuples** : a sequence of data elements $\langle \vec{d} \rangle$.
- **Relations** or tables: finite sets of records (of the same width)
- Database (**instance**): collection of named relations
- **Schema**: a description of the structure of a record, table, or database

Schemas

- Relations "typed" by number of arguments: $R{:}n$ iff $R \subset D^n$
- The only base type is D, so number of arguments suffices. If $R{:}n$, define **arity** $\mathrm{ary}(R) = n$
- Database instances `I` are "typed" by schemas $\mathcal{R}$

$$\mathcal{R} ::= \{R_1{:}m_1, \ldots, R_n{:}m_n\}$$

- Example:

$$R{:}2, S{:}3$$

  means "relation R has two fields, and S has three fields."

Instances

An instance $\mathtt{I}$ (of a database schema $\mathcal{R}$) is a collection of finite relations matching $\mathcal{R}$

$$\Big\{R = \{\langle a, b\rangle, \langle b, c\rangle\}, S = \{\langle a, a, a\rangle, \langle a, b, b\rangle\}\Big\}$$

is an instance of $\mathcal{R} = \{R{:}2, S{:}3\}$

- Order of rows doesn't matter, duplicates irrelevant
- $\mathrm{adom}(R) \subset D$ is the **active domain** of R, i.e. the set of domain values that occur in R
- Tabular notation:

R

a	b
b	c

S

a	a	a
a	b	b

Exercises 1/2

1. Definitions

Database, Table, Relation, Attribute, Tuple, Schema, Instance, Domain, Value, Active domain

2. True or False?

i) A relation may have an infinite number of tuples.
ii) Every relation follows a schema.
iii) Attributes are typed.
iv) $\mathtt{I} \subset \mathtt{J}$ implies that $\text{adom}(\mathtt{I}) \subset \text{adom}(\mathtt{J})$.

Exercises 2/2

3. Is it a relation?

Which of the following relations conform to the schema $R{:}2$?

1. $R = \{\langle a, b\rangle, \langle c, b\rangle\}$
2. $R = \{\langle a, b\rangle, \langle c\rangle\}$
3. $R = \{\langle a, b\rangle, \langle b, a\rangle\}$
4. $R = \{\langle a, b\rangle, \langle a, b\rangle\}$ (abuse of set notation)
5. $R = \{\langle a, b\rangle, \langle c, \mathrm{NULL}\rangle\}$
6. $R = \{\} = \emptyset$
7. $R = \{\langle\rangle\}$

4. Problem

How many different ways are there to draw a relation of schema $S{:}n$ within m tuples?

Core algebra

Expressions:

$$
\begin{array}{rcl}
v & ::= & i \mid a \\
Q & ::= & \langle a \rangle \mid R \mid \sigma_{v=w}(Q) \mid \pi_{\vec{i}}(Q) \mid Q \times Q'
\end{array}
$$

- Index i, domain value a
- Singleton (1-tuple constant) $\langle a \rangle$
- Relation variables R
- **S**election σ
- **P**rojection π
- **C**ross product x

This core algebra is so-called **SPC**

Selection

- $\sigma_{v=w}(Q)$ selects those rows $\langle \vec{d} \rangle$ satisfying $v = w$ from Q
- Here, v, w are either indices or domain constants
- Example:

$$\sigma_{1=2}\left(\begin{array}{|c|c|c|} \hline \color{red}{a} & \color{red}{a} & b \\ a & b & c \\ \color{red}{b} & \color{red}{b} & d \\ b & c & c \\ \color{red}{c} & \color{red}{c} & d \\ \hline \end{array}\right) = \begin{array}{|c|c|c|} \hline a & a & b \\ b & b & d \\ c & c & d \\ \hline \end{array}$$

Projection

- $\pi_{\vec{i}}(Q)$ projects the field indices listed in $\vec{i}$, in order (that is, drops all other fields)
- Example:

$$\pi_{1,3}\left(\begin{array}{|c|c|c|} \text{a} & \text{a} & \text{b} \\ \text{a} & \text{b} & \text{b} \\ \text{b} & \text{b} & \text{d} \\ \text{b} & \text{c} & \text{c} \\ \text{c} & \text{c} & \text{d} \\ \hline \end{array}\right) = \begin{array}{|c|c|} \hline \text{a} & \text{b} \\ \text{b} & \text{d} \\ \text{b} & \text{c} \\ \text{c} & \text{d} \\ \hline \end{array}$$

- Set-theoretic semantics: **duplicate elimination**

Cross product

- $Q \times Q'$ is the cross product of Q and Q'
- Example:

Q

a	a	b
a	b	c
b	b	d

$\times$

Q'

a	55
c	42

$=$

$Q \times Q'$

a	a	b	a	55
a	a	b	c	42
a	b	c	a	55
a	b	c	c	42
b	b	d	a	55
b	b	d	c	42

Normalization

- Two queries are **equivalent** ($Q_1 \equiv Q_2$) iff for every input, they produce the same output
- There are obviously lots of ways of writing "the same" query
- It would be handy to have a **canonical** representation of a given query

Theorem

Every SPC query has a normal form such that $Q \equiv Q'$ if and only if $norm(Q) = norm(Q')$

General idea: Rewrite to form

$$\pi_{\vec{j}}\Big(\sigma_F(R_1 \times \cdots \times R_n)\Big)$$

Normalizing selection and projection

- Neighboring projections can be composed

$$\pi_{\vec{j}}(\pi_{\vec{k}}(Q)) = \pi_{\vec{\ell}}(Q)$$

  where $\ell_i = k_{j_i}$ for each i

- Problem: Neighboring selections commute

$$\sigma_{i=j}(\sigma_{i'=j'}(Q)) = \sigma_{i'=j'}(\sigma_{i=j}(Q))$$

- Solution: Allow **sets** (conjunction) of equations in selections

$$\sigma_F(\sigma_{F'}(Q)) = \sigma_{F \cup F'}(Q)$$

More rewriting rules

$$
\begin{array}{rcl}
\sigma_F(\pi_{\vec{j}}(Q)) & = & \pi_{\vec{j}}(\sigma_{F'}(Q)) \\
 & & \text{where } F' = F[j_i/i] \\
\sigma_{1=j}(\langle a \rangle \times Q) & = & \langle a \rangle \times \sigma_{(j-1)=a}(Q) \\
((Q_1 \times \cdots \times Q_n) \times Q) & = & (Q_1 \times \cdots \times Q_n \times Q) \\
(Q \times (Q_1 \times \cdots \times Q_n)) & = & (Q \times Q_1 \times \cdots \times Q_n) \\
Q \times Q' & = & \pi_{\vec{\ell}\vec{m}}(Q' \times Q) \\
 & & \text{where } \vec{m} = 1, \ldots, \mathrm{ary}(Q'), \\
 & & \vec{\ell} = \mathrm{ary}(Q') + 1, \ldots, \mathrm{ary}(Q) + \mathrm{ary}(Q') \\
\sigma_F(Q) \times Q' & = & \sigma_F(Q \times Q') \\
\pi_{\vec{\ell}}(Q) \times Q' & = & \pi_{\vec{\ell}}(Q \times Q')
\end{array}
$$

Numbers vs. names

- So far, we have used indices to refer to values in records
- This is convenient from a theoretical perspective because such expressions are concise and easy to specify
- But in real life (e.g. SQL), **field names** are useful
- Example. Guess what this does:

$$\pi_{1,2,5}(\sigma_{1=4}(R \times S))$$

How about this?

$$\pi_{\text{Name,Address,Phone}}\left(\sigma_{Name=PName}(\text{AddressDir} \times \text{PhoneDir})\right)$$

Schemas with names

- Relations $R{:}\langle\vec{A}\rangle$ are typed by list of field names
- We say **sort** of R is $\langle\vec{A}\rangle$
- The only base type remains D, so list of names suffices
- Database instances are typed by schemas $\mathcal{R}$ mapping relation names to sorts
- Example:

$$\mathcal{R} = \{R{:}\langle A, B\rangle,\ S{:}\langle A, B, C\rangle\}$$

  means "relation R has two fields named A, B, and relation S has three fields named A, B, C."

A running example

Cinema database

```
Cinema = {Movie, Featuring, Location, Schedule}

Movie:     title, director, length, release_date
Featuring: title, actor, role
Location:  theater, address, phone_number
Schedule:  theater, title, showtime
```

Instances with names

- A named record is a finite map $\langle A_1{:}d_1, \ldots, A_n{:}d_n\rangle$ from names to values
- By convention, field order doesn't matter
- A named relation $R{:}\langle\vec{A}\rangle$ is a set of named records
- An instance of $\mathcal{R}$ is a collection of named relations matching named schema $\mathcal{R}$
- An instance of $\mathcal{R} = \{R{:}\langle A, B\rangle,\ S{:}\langle A, B, C\rangle\}$:

R

A	B
a	a
b	c
c	c

S

A	B	C
a	a	a
b	d	e
c	e	f

Core algebra with names

$$
\begin{array}{rcl}
v & ::= & A \mid a \\
Q & ::= & \langle A{:}a \rangle \mid R \mid \sigma_{v=w}(Q) \mid \pi_{\vec{A}}(Q) \mid Q \bowtie Q' \mid \rho_{A_1 \ldots A_n \to B_1 \ldots B_n}(Q)
\end{array}
$$

- Singleton constants $\langle A{:}a \rangle$, relation variables R
- **S**election σ
- **P**rojection π
- **J**oin $\bowtie$
- **R**enaming ρ

This algebra is called **SPJR**

Selection

- $\sigma_{v=w}(Q)$ selects those rows satisfying $v = w$ from Q
- Here, v, w are either constants or implicit field labels
- Example:

$$\sigma_{A=B}\left(\begin{array}{|c|c|c|}\hline A & B & C \\ \hline a & a & b \\ a & b & c \\ b & b & d \\ b & c & c \\ c & c & d \\ \hline\end{array}\right) = \begin{array}{|c|c|c|}\hline A & B & C \\ \hline a & a & b \\ b & b & d \\ c & c & d \\ \hline\end{array}$$

Projection

- $\pi_{\vec{A}}(Q)$ projects the field names listed in $\vec{A}$ (that is, drops all other fields)
- Example:

$$\pi_{A,C}\left(\begin{array}{|c|c|c|}\hline A & B & C \\ \hline a & a & b \\ a & b & b \\ b & b & d \\ b & c & c \\ c & c & d \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline A & C \\ \hline a & b \\ b & d \\ b & c \\ c & d \\ \hline\end{array}$$

Join

- $Q \bowtie Q'$ joins Q and Q' by merging all pairs of records whose **common fields are equal**. Duplicate fields are omitted
- Example:

A	B	C
a	a	b
a	b	c
b	b	d
b	c	c
c	c	d

$\bowtie$

A	D
a	55
a	100
c	42

$=$

A	B	C	D
a	a	b	55
a	a	b	100
a	b	c	55
a	b	c	100
c	c	d	42

Renaming

- The renaming operator $\rho_{A_1 \dots A_n \to B_1 \dots B_n}(R)$ applies the simultaneous substitution $[B_1/A_1, \dots, B_n/A_n]$ to the field names of R.
- Example:

$$\rho_{BC \to DB}\left(\begin{array}{|c|c|c|}\hline A & B & C \\ \hline a & a & b \\ a & b & c \\ b & b & d \\ b & c & c \\ c & c & d \\ \hline\end{array}\right) = \begin{array}{|c|c|c|}\hline A & D & B \\ \hline a & a & b \\ a & b & c \\ b & b & d \\ b & c & c \\ c & c & d \\ \hline\end{array}$$

Equivalence

- Unnamed and named relations are equivalent
- Named [records, relations, instances, schemas] are in **bijective correspondence** with unnamed [records, relations, instances, schemas]
- Howto: fix an ordering on field names; assume all field name sequences are in increasing order

$$\langle d_1, \ldots, d_n \rangle{:}n \Leftrightarrow \langle A_1{:}d_1, \ldots, A_n{:}d_n \rangle{:}\langle \vec{A} \rangle$$

Equivalence

- A SPJR query $Q{:}\langle\vec{A}\rangle$ on $\mathcal{R}$ defines a function from named instances $\mathtt{I}$ of $\mathcal{R}$ to named relations $Q(\mathtt{I})$ over $\langle\vec{A}\rangle$
- We say that two query languages are **equivalent** if their expressions define the same functions

Theorem

SPC and SPJR queries are equivalent in expressive power

- This is good because SPC is somewhat "lower level": easier to analyze, but not as convenient
- Hence, can "compile" SPJR queries down to SPC queries without loss of expressiveness

Equivalence: sketch of the proof

It suffices to translate SPC queries to SPJR queries and *vice-versa*

- Selection, projection, renaming cases are easy
- **Key case 1**: $(R \bowtie S)^* = \pi_{\vec{i}}(\sigma_{i_1=j_1,\ldots,i_n=j_n}(R^* \times S^*))$ where i_1, j_1, etc. are indices of equal field names in R and S
- **Key case 2**: $(R^* \times S^*) = (\rho_{A_1\ldots A_n \to B_1\ldots B_n}(R)) \bowtie S$ where $\langle \vec{B} \rangle$ is all distinct from S field names
- Careful proof/implementation requires **bookkeeping** (to maintain mapping between field names and positions)

Exercises 1/2

1. Definitions

Algebra, SPC, SPJR, Query normal form, Query equivalence

2. True or False?

i) Renaming is only "syntactic sugar" in SPJR.

ii) Every SPC query is expressible within SPJR and *vice-versa*.

iii) $\pi_{\vec{j}}(\pi_{\vec{k}}(R)) = \pi_{\vec{k}}(\pi_{\vec{j}}(R))$.

iv) $\{\}$ op $R = R, \text{op} \in \{\times, \bowtie\}$.

v) $\{\langle\rangle\}$ op $R = R, \text{op} \in \{\times, \bowtie\}$.

vi) $\sigma_{i=a \,\vee\, j=b}(R)$ is expressible within SPC.

vii) $\sigma_{i \neq a}(R)$ is not expressible within SPC.

Exercises 2/2

3. SPC and SPJR

1. Give the all schedule of **Katorza** theater.
2. Find theaters that show some movies directed by **Chabrol**.
3. Give the addresses of theaters that play some movies featuring **Jaoui** and where the director is also an actor.

4. Problem

Let $\mathcal{R}$ be a database schema and Q a SPC query over $\mathcal{R}$;

1. Prove that $Q(\mathtt{I})$ is finite for each instance $\mathtt{I}$ of $\mathcal{R}$.
2. Given instance $\mathtt{I}$ of $\mathcal{R}$ and output arity n for SPC query $Q(\mathtt{I})$:n, show an upper bound for the number of tuples that can occur in $Q(\mathtt{I})$. Show that this bound can be achieved.

Conjunctive calculus

- Calculus is a query formalism based on **set comprehension** notation
- Fragment of **first-order logic** (FO)

$$
\begin{array}{rcl}
v & ::= & a \mid x \\
\Phi & ::= & R(\vec{v}) \mid v = v' \mid \exists x.\Phi \mid \Phi \wedge \Psi \\
Q & ::= & \{(\vec{v}) \mid \Phi\}
\end{array}
$$

- Because only conjunctions are allowed in Φ, these are called conjunctive queries

Evaluation of formula

Semantics

Returns all tuples $\vec{v}$ such that Φ is true

- **Free variables** free(Φ) are those that occur in $\vec{v}$
- adom(Φ) is the **active domain** of Φ, i.e. set of constants in Φ
- Valuation ν maps variables to constants from D: $\nu(\vec{x}) = \vec{a}$
- $\mathtt{I}$ satisfies Φ under ν denoted $\mathtt{I} \models \Phi[\nu]$

$$Q(\mathtt{I}) = \{\nu(\vec{v}) \mid \mathtt{I} \models \Phi[\nu] \text{ and } \nu \text{ is a valuation over } \vec{v}\}$$

Calculus example

$$\{(x, y, z) \mid \exists w. R(x, y, w) \wedge S(x, z)\}$$

$$\overset{R}{\begin{array}{|c|c|c|}\hline a & a & b \\ a & b & c \\ b & b & c \\ c & b & d \\ \hline\end{array}}\;,\;\overset{S}{\begin{array}{|c|c|}\hline a & 55 \\ c & 42 \\ \hline\end{array}} = \overset{\text{Ans}}{\begin{array}{|c|c|c|}\hline a & a & 55 \\ a & b & 55 \\ c & b & 42 \\ \hline\end{array}}$$

- w could be an **anonymous variable**:

$$\{(x, y, z) \mid R(x, y, \_) \wedge S(x, z)\}$$

More examples

Find addresses of theaters that play movies directed by Chabrol

$$
\begin{aligned}
Q \quad = \quad & \{(x, y) \mid \exists z.\mathrm{Movie}(z, \mathit{Chabrol}, \_, \_) \wedge \\
& \mathit{Location}(x, y, \_) \wedge \mathit{Schedule}(x, z, \_)\}
\end{aligned}
$$

Expressiveness

- Selection:

$$\sigma_{v=w}(R) = \{(\vec{x}) \mid R(\vec{x}) \wedge \vec{x}_{[v]} = \vec{x}_{[w]}\}$$

  where $\vec{x}_{[i]} = x_i, \vec{x}_{[a]} = a$

- Projection:

$$\pi_{\vec{\ell}}(R) = \{(x_{\ell_1}, \ldots, x_{\ell_k}) \mid R(\vec{x})\}$$

- Cross product:

$$R \times S = \{(\vec{x}, \vec{y}) \mid R(\vec{x}) \wedge S(\vec{y})\}$$

Tableau queries

- $(\mathbf{T}, \vec{u})$: it consists of a set of tables, so-called **Tableau T** with constant and variable entries, and an answer row $\vec{u}$

$$
\begin{array}{c}
\begin{array}{|c|c|c|}
\hline
\multicolumn{3}{|c|}{R} \\
\hline
v_{11} & \cdots & v_{1n} \\
\hline
\vdots & \vdots & \vdots \\
\hline
v_{i1} & \cdots & v_{in} \\
\hline
\end{array} \\
\vdots \\
\begin{array}{|c|c|c|}
\hline
\multicolumn{3}{|c|}{S} \\
\hline
w_{11} & \cdots & w_{1m} \\
\hline
\vdots & \vdots & \vdots \\
\hline
w_{j1} & \cdots & w_{jm} \\
\hline
\end{array}
\end{array}
\Longrightarrow
\begin{array}{|c|c|c|}
\hline
x_1 & \cdots & x_p \\
\hline
\end{array}
$$

Tableau query: example

Find addresses of theaters that play movies directed by Chabrol

Movie			
x_{ti}	Chabrol	x_{le}	x_{rd}

Location		
x_{th}	x_{ad}	x_{pn}

Schedule		
x_{th}	x_{ti}	x_{st}

$$\Longrightarrow \langle x_{th}, x_{ad} \rangle$$

Expressiveness

- Selection $\sigma_{i=j}(R)$:

$$\begin{array}{|c|c|c|}\hline \multicolumn{3}{|c|}{R} \\ \hline v_1 & \cdots & v_n \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline v_1 & \cdots & v_n \\ \hline \end{array}$$

  where $v_i = v_j = y$, $v_k = x_k$ otherwise

- Selection $\sigma_{i=a}(R)$:

$$\begin{array}{|c|c|c|}\hline \multicolumn{3}{|c|}{R} \\ \hline v_1 & \cdots & v_n \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline v_1 & \cdots & v_n \\ \hline \end{array}$$

  where $v_i = a$, $v_k = x_k$ otherwise

Expressiveness (cont'd)

- Projection $\pi_{\vec{\ell}}(R)$:

$$\begin{array}{|c|c|c|}\hline \multicolumn{3}{|c|}{R} \\ \hline x_1 & \cdots & x_n \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|c|}\hline x_{\ell_1} & \cdots & x_{\ell_n} \\ \hline \end{array}$$

- Cross product $R \times S$:

$$\begin{array}{c} \begin{array}{|c|c|c|}\hline \multicolumn{3}{|c|}{R} \\ \hline x_1 & \cdots & x_n \\ \hline \end{array} \\ \\ \begin{array}{|c|c|c|}\hline \multicolumn{3}{|c|}{S} \\ \hline y_1 & \cdots & y_m \\ \hline \end{array} \end{array} \Longrightarrow \begin{array}{|c|c|c|c|c|c|}\hline x_1 & \cdots & x_n & y_1 & \cdots & y_m \\ \hline \end{array}$$

Conjunctive queries

So far, we've seen several **equivalent** query languages:

- SPC
- SPJR
- Conjunctive Calculus
- Tableau

These are all conjunctive query languages

Why is it so important?

Property 1

Conjunctive queries are closed under composition

For all CQs $\mathcal{Q}' \vdash Q$ and $\mathcal{R} \vdash Q'$, then $Q \circ Q'$ is CQ as well

Property 2

Conjunctive queries are satisfiable

There exists an instance $\mathtt{I}$ of $\mathcal{R}$ for which $Q(\mathtt{I})$ is non-empty

Property 3

Conjunctive queries are monotonic

For all instances $\mathtt{I}$, $\mathtt{J}$ over $\mathcal{R}$ such that $\mathtt{I} \subseteq \mathtt{J}$, then $Q(\mathtt{I}) \subseteq Q(\mathtt{J})$

About satisfiability

- Conjunctive calculus with **equality** raises the following problem:

$$\{(\vec{x}) \mid R(\vec{x}) \wedge x_i = a \wedge x_i = b\} = Q^{\emptyset}$$

- **Unsatisfiable queries** could easily be checked by transitive closure of equality terms
- SPC and SPJR include unsatisfiable queries as well:
  - $\sigma_{\{i=a,i=b\}}(R) = Q^{\emptyset}$
  - $\sigma_{\{A=a,A=b\}}(R) = Q^{\emptyset}$

CQs are satisfiable queries only

Why is it so important? (cont'd)

Property 4

Conjunctive query containment is decidable and complexity is **NP-complete**

$\mathcal{R} \vdash Q \subseteq \mathcal{R} \vdash Q'$ iff for each instance $\mathtt{I}$ of $\mathcal{R}$, $Q(\mathtt{I}) \subseteq Q'(\mathtt{I})$

- Small Q on large DB makes it acceptable
- Equivalence by checking mutual containment
  $Q \subseteq Q' \wedge Q \supseteq Q'$
- Very useful for query evaluation and optimization

Exercises 1/2

1. Definitions

Conjunctive query, Calculus, Tableau query, Satisfiability, Monotonicity

2. True or False?

i) Conjunctive calculus is a fragment of FO.
ii) SPC expressions are satisfiable queries only.
iii) SPC and SPJR algebras are equivalent to CQs.
iv) Conjunctive calculus w/o equality is no more CQ.
v) NP-completeness of CQ query containment is a mess.
vi) $\{R(1,2)\}$ is a valid calculus formula.

Exercises 2/2

3. Conjunctive Calculus and Tableau Queries
   1. Write calculus formulas for queries of the previous section.
   2. Draw an unsatisfiable Tableau query.

4. Problem✐
   1. Give evidence of satisfiability of CQs.
   2. Give evidence of monotonicity of CQs.

What can't conjunctive queries handle?

Lots!

- Union $\cup$ / disjunction $\vee$
- Difference $-$ / negation $\neg$ and universal quantification $\forall$
- Recursive queries (e.g., transitive closure, connectivity)
- Primitive operations on D (arithmetic, string operations)
- Aggregation: counting, sum, average
- Complex data (nested records, variants, arbitrary trees)
- Arbitrary computations

Set operations

Relations are sets of tuples

So what about set operations?

- Intersection $\cap$
- Union $\cup$
- Complement or set difference $-$

Actually...

- Intersection is **already expressible** in SPC
- Union, complement are not
- How would one show this?
  Idea: Follows from normal form theorem
- In fact, the value $\{\langle a \rangle, \langle b \rangle\}$ is not even expressible in SPC

Core algebra with union

$$
\begin{aligned}
v &::= i \mid a \\
Q &::= \langle a \rangle \mid R \mid \sigma_{v=w}(Q) \mid \pi_{\vec{i}}(Q) \mid Q \times Q' \mid \mathbf{Q} \cup \mathbf{Q}'
\end{aligned}
$$

- **S**election σ
- **P**rojection π
- **C**ross product $\times$
- **U**nion $\cup$

This calculus is called **SPCU**

Named core algebra with union

$$
\begin{array}{rcl}
v & ::= & A \mid a \\
Q & ::= & \langle A{:}a \rangle \mid R \mid \sigma_{v=w}(Q) \mid \pi_{\vec{A}}(Q) \\
 & & \mid Q \bowtie Q' \mid \rho_{A_1 \ldots A_n \to B_1 \ldots B_n}(Q) \mid \mathbf{Q} \cup \mathbf{Q}'
\end{array}
$$

- **S**election σ
- **P**rojection π
- **J**oin $\bowtie$
- **R**enaming ρ
- **U**nion $\cup$

This calculus is called **SPJRU**

Difference

- Difference $Q - Q'$ can easily be added to [SPCU, SPJRU]
- [SPCU, SPJRU] $+$ difference are called the (named)

Relational Algebra

- Union/disjunction and difference/negation pose little or no difficulty for algebra

Theorem

Query equivalence is **no more decidable** in RA

So far so good!

Add syntax for disjunction and negation to **relational calculus**:

$$\begin{array}{rcl} \Phi & ::= & \exists x.\Phi \mid R(\vec{v}) \mid v = v' \mid \Phi \vee \Psi \mid \Phi \wedge \Psi \mid \neg\Phi \\ Q & ::= & \{(\vec{v}) \mid \Phi\} \end{array}$$

- Really, Φ can be an arbitrary first-order formula since $\forall x.\Phi \equiv \neg\exists x.\neg\Phi$
- Very really, $\wedge$, $\exists$, $\neg$ suffice since $\Phi \vee \Psi \equiv \neg(\neg\Phi \wedge \neg\Psi)$
- **Semantics**: same as for ordinary first-order (FO) formulas?

Disjunction: problems

Conjunctive calculus + disjunction = **positive calculus**
Positive calculus is equivalent to [SPJRU, SPCU]

$$\{(x, y, z) \mid R(x, y) \vee R(y, z)\}$$

- For nonempty R, **answer is infinite** due to free x or z
- Unsafe query may produce infinite output from finite input
- Because databases are finite, we restrict to **safe queries**

Negation: problems

- Give it a try:

$$\{(x) \mid \neg R(x)\}$$

- Obviously unsafe... and this one:

$$\{(x) \mid \forall y.R(x,y)\} = \{(x) \mid \neg \exists y.\neg R(x,y)\}$$

- Whoa! This query is **domain-dependent**: answer depends on domain of quantification
  - e.g. if $R(a,b)$, $R(a,a)$ hold, and y ranges over $\{a,b\}$, then $x=a$ works...
  - but if y ranges over $\{a,b,c\}$ then $x=a$ doesn't work!

Safe domain-independent queries

Wanted: Guarantees that FO makes sense as a query language

- Unfortunately, **domain independence is undecidable**
- Two solutions:
  1. Find decidable sufficient conditions for domain dependence; prove that no loss of expressiveness ensues
  2. Observe that infinite models have finite descriptions in terms of equations; generalize database theory to finite sets of constraints rather than finite sets of facts
- Approach #2 is more general/satisfying; usually, focus on #1

Safety analysis

Range-restricted variable $x \in \text{free}(\Phi)$ s.t. $\nu(x) \in \text{adom}(\Phi, \mathtt{I})$
Basic idea: every head variable must be constrained in the body

$$
\begin{array}{rcl}
rr(R(\vec{v})) & = & \text{free}(R(\vec{v})) \\
rr(x = a) = rr(a = x) & = & \{x\} \\
rr(\Phi \wedge x = y) & = & \begin{cases} rr(\Phi) & (x, y \notin rr(\Phi)) \\ rr(\Phi) \cup \{x, y\} & (x \in rr(\Phi) \text{ or } y \in rr(\Phi)) \end{cases} \\
rr(\Phi \wedge \Psi) & = & rr(\Phi) \cup rr(\Psi) \\
rr(\Phi \vee \Psi) & = & rr(\Phi) \cap rr(\Psi) \\
rr(\neg \Phi) & = & \emptyset \\
rr(\exists x.\Phi) & = & \begin{cases} rr(\Phi) - \{x\} & (x \in rr(\Phi)) \\ \bot & (x \notin rr(\Phi)) \end{cases}
\end{array}
$$

$\{\vec{v} \mid \Phi\}$ is **safe-range** iff for some $\Psi \equiv \Phi$, then $\text{free}(\vec{v}) \subseteq rr(\Psi)$

Exercises 1/2

1. Definitions

Relational algebra, Positive calculus, Safe query,
Domain-dependent query, Safe-range query

2. True or False?

i) Union and difference make CQ become RA.
ii) Safe-range RC is equivalent to RA.
iii) Query containment can be evaluated algorithmically.
iv) It is possible to express unsafe query in RA.
v) Domain-dependancy is decidable.
vi) $\{(x, y, z) \mid R(x, y) \vee R(y, z)\}$ is finite when x, y, z in adom(R).

Exercises 2/2

3. RA and RC ✐

1. Which theaters do not show any movies directed by **Chabrol**?
2. Which theaters show only movies directed by **Chabrol**?
3. Which theaters show all movies directed by **Chabrol**?

4. Problem

For each of the following queries, guess whether it is domain-independent and/or safe range.

$$\{(x, y) \mid \exists z.(R(x, z) \wedge \exists w.S(w, x, y)) \wedge x = y\} \tag{1}$$

$$\{\langle\rangle \mid \exists x \forall y.(R(y) \rightarrow S(x, y))\} \tag{2}$$

$$\{(x, y) \mid (x = a \vee \exists z.R(y, z)) \wedge S(y)\} \tag{3}$$

If it is not domain-independent, exhibit a counter-example; and if it is safe range, translate it into an RA expression.

Declarative vs. Procedural

- Query languages are declarative: **what** in the output?

$$\{x_{th} \mid \exists x_{ti}.\texttt{Movie}(x_{ti}, \text{Chabrol}, \_, \_) \wedge \texttt{Schedule}(x_{th}, x_{ti}, \_)\}$$

- Database system operates internally with different, procedural languages, which specify how to get the result

```
for each tuple T1=(ti1,di,le,rd) in table Movie do
  for each tuple T2=(th,ti2,ts) in table Schedule do
    if ti1=ti2 and di='Chabrol' then output th
  end
end
```

Declarative vs. Procedural (cont'd)

Theoretical languages

- Declarative: relational calculus
- Procedural: relational algebra

Practical languages

Mix of both but mostly one use declarative features

- QBE (Tableau queries)
- SQL

SQL

- Structured Query Language
- Developed originally at IBM in the late 70s
- First standard: SQL-86 (with minor revision in SQL-89)
- Second standard: SQL-92
- Latest standard: SQL-99, or **SQL3**, well over 1,000 pages
- SQL:2003 and SQL:2008 add extra features
- De-facto standard of the relational database world—replaced all other languages

Example of SQL queries

Reminder of the database schema:

```
Movie:     title, director, length, release_date
Featuring: title, actor, role
Location:  theater, address, phone_number
Schedule:  theater, title, showtime
```

Find titles of current movies

```
SELECT Title
FROM Movie ;
```

- `SELECT` lists attributes that go into **the output** of a query
- `FROM` lists **input relations**
- Algebraic formula: $\pi_{\text{title}}(\text{Movie})$
- Warning: SQL uses **bags** rather than sets

Relational operations on bags

RDBMS implements relations as bags (or multisets)

- Union adds occurrences: $|R \cup S| = |R| + |S|$
- Duplicates are preserved in projection: $|\pi_{\vec{A}}(R)| = |R|$
- Consistent aggregate computation: AVG(R[$\vec{A}$])
- t in $R \cap S$: $\min(|R(t)|, |S(t)|)$
- t in $R - S$: $\max(0, |R(t)| - |S(t)|)$
- No surprise for σ, $\times$, $\bowtie$
- Algebraic laws revisited: $(R \cup S) - T = (R - T) \cup (S - T)$?

Relations as multisets

The results for Conjunctive Query containment over relations-as-sets no longer hold!

Property 1

CQ containment is no longer in NP

Property 2

Query containment of unions of CQ is undecidable

Lesson:

Small changes in the data model can have a big impact

Example of SQL queries (cont'd)

Find theaters showing movies directed by Chabrol

```sql
SELECT S.Theater
FROM Schedule S, Movie M
WHERE M.Title = S.Title
     AND M.Director='Chabrol' ;
```

New features:

- `SELECT` now specifies which relation the attributes came from—because we use more than one
- `FROM` lists two relations with aliases
- `WHERE` specifies the **condition** for selecting a tuple

Translation to SPC

Conjunctive Query

`SELECT DISTINCT-FROM-WHERE` SQL queries with conjunct of equality conditions are equivalent to SPC

- Algebraic expression:

$$\underbrace{\pi_{\text{S.Theater}}}_{\texttt{SELECT statement}}\ (\underbrace{\sigma_{\text{M.Title=S.Title}\wedge\text{M.Director=''Chabrol''}}}_{\texttt{WHERE statement}}(\underbrace{\text{Schedule S} \times \text{Movie M}}_{\texttt{FROM statement}}))$$

SQL and RC

Tuple Relational Calculus

Core declarative part of SQL is close to TRC, a variant of Relational Calculus

$$\{t.\text{theater} \mid \exists u.\text{Movie}(u) \wedge \textit{Schedule}(t) \wedge \\ t.\text{title} = u.\text{title} \wedge u.\text{director} =' \text{Chabrol}'\}$$

- Same operators than (Domain)RC
- Variables are **tuples**
- Attribute values can be reached b.t.w. of dot notation

TRC is obviously equivalent to (D)RC

Joining relations

- `WHERE` allows to join together several relations
  `R.Title = S.Title`
- A `JOIN` statement also exists in SQL

```
SELECT S.Theater
FROM Movie M NATURAL JOIN Schedule S
WHERE M.Director='Chabrol' ;
```

Join variants

Natural join is the $\bowtie$ operator from SPJR

- Preferred SQL statement:

```
SELECT S.Theater FROM Movie M
  INNER JOIN Schedule S USING (Title)
  WHERE M.Director='Chabrol' ;
```

- Family of join operators in SQL:
  - **Equi-join**: `R INNER JOIN S ON R.A=S.B`
  - θ**-join**: `R INNER JOIN S ON R.A > S.B`
  - **Outer join**: `R LEFT OUTER JOIN S ON R.A != S.B`
  - **Cross product**: `R CROSS JOIN S`

Positive Relational Algebra within SQL

Reminder

SPJR+Union operator makes Positive RA

Find actors who played in movies directed by Chabrol *OR* Polanski

```
SELECT F.Actor FROM Featuring F JOIN Movie M USING (Title)
  WHERE M.Director='Chabrol' OR M.Director='Polanski' ;
```

SQL has also a dedicated `UNION` operator

```
  (SELECT Actor FROM Featuring JOIN Movie USING (Title)
  WHERE Director='Chabrol')
UNION
  (SELECT Actor FROM Featuring JOIN Movie USING (Title)
  WHERE Director='Polanski') ;
```

More Thoughts on Union

- `UNION ALL` to **preserve duplicates**
- **Renaming** to the rescue

List all directors or actors

```
  (SELECT Director AS Person FROM Movie)
UNION
  (SELECT Actor AS Person FROM Featuring) ;
```

Intersection and Difference

$\cap$ is expressible within SPJRU, $-$ is not

$$R \cap S = \rho_{\mathrm{sort}(R) \to \vec{X}}(R) \bowtie \rho_{\mathrm{sort}(S) \to \vec{X}}(S)$$

SQL syntax for

- $R \cap S$: `R INTERSECT S` or `R INTERSECT ALL S`
- $R - S$: `R EXCEPT S` or `R EXCEPT ALL S`

Find all actors who are...

not directors

```
  (SELECT Actor AS Person
   FROM Featuring)
EXCEPT
  (SELECT Director AS Person
   FROM Movie) ;
```

also directors

```
  (SELECT Actor AS Person
   FROM Featuring)
INTERSECT
  (SELECT Director AS Person
   FROM Movie) ;
```

Beyond simple queries

- So far we mostly translated RA=$\{\sigma, \pi, \bowtie, \rho, \cup, -\}$ into SQL
- Other SQL statements allow to express complex queries in a slightly different way than pure RA
  - Nested queries
  - "For all" queries
- Also, extra SQL features go far beyond RA

Nested queries

- WHERE clause could contain **subquery**

Find actors who did not play in a movie by Chabrol

```
SELECT F.Actor FROM Featuring F
WHERE F.Actor NOT IN (SELECT F1.Actor FROM Featuring F1
              JOIN Movie M USING (Title)
              WHERE M.Director='Chabrol') ;
```

- Nested query could be **correlated** to the outermost query

Find dirs. whose movies are playing at Le Katorza

```
SELECT M.Director FROM Movie M
WHERE EXISTS (SELECT * FROM Schedule S JOIN M USING (Title)
              WHERE S.Theater='Le Katorza') ;
```

- Actually, nested query could occur anywhere!

For all in SQL

Find directors whose movies are playing in all theaters

$$\{(x.\text{director}) \mid M(x) \ \wedge \ \forall y[L(y), \exists z(S(z) \wedge \\ z.\text{title} = x.\text{title} \ \wedge \ y.\text{theater} = z.\text{theater})]\}$$

- M stands for Movie and S for Schedule and L for Location

$$\pi_{\text{director}}(M) - \pi_{\text{director}}\Big((\pi_{\text{theater}}(L) \times \pi_{\text{director}}(M)) - \pi_{\text{theater,director}}(M \bowtie S)\Big)$$

- RA query is much less intuitive than RC query

For all in SQL (cont'd)

SQL's way of saying this

Find directors such that **there does not exist** a theater where their movies **do not** play

- Main idea: $\forall x.P(x) \Leftrightarrow \neg\exists x.\neg P(x)$

```
SELECT M.Director FROM Movie M
WHERE NOT EXISTS (SELECT * FROM Location L
        WHERE NOT EXISTS (SELECT * FROM Movie M1
                          JOIN Schedule S ON (Title)
                          WHERE M1.Director=M.Director
                          AND S.Theater=L.Theater)) ;
```

Relational division

$$\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}\Big((\pi_{A-B}(\alpha) \times \beta) - \alpha\Big)$$

At least four ways to write division in SQL

1. Direct conversion of the relational algebra expression: translate $-$, π, $\times$
2. Logical tautology (see previous slide)
3. Set containment: if $X \supseteq Y$, then $Y - X = \emptyset$ (or $\neg\exists x.x \in Y - X$)
4. Set cardinalities: check for $|Y| - |X| = 0$. Need for SQL group by/having and count, discard negation

Relational division (cont'd)

$$\pi_{\text{theater,director}}(\text{Movie} \bowtie \text{Schedule}) \div \pi_{\text{theater}}(\text{Location})$$

<set containment version>

```
SELECT M.Director FROM Movie M
WHERE NOT EXISTS (  (SELECT L.Theater FROM Location L)
                  EXCEPT
                    (SELECT S.Theater FROM Schedule S
                     JOIN Movie M1 USING (Title)
                     WHERE M1.Director=M.Director) ) ;
```

Other features of SQL

- Datatypes, type-specific operations
- Table declaration, constraint enforcement (DDL part)
- Database modifications: insert, update, delete
- NULL values
- Views and temporary tables
- Aggregation

Simple aggregate queries

Count the number of Movies

```
SELECT COUNT(*)
FROM Movie ;
```

Add up all movie lengths

```
SELECT SUM(Length)
FROM Movie ;
```

Find the number of directors

```
SELECT COUNT(DISTINCT Director)
FROM Movie ;
```

Aggregation and grouping

For each theater playing at least one long (over 2 hours) movie, find the average length of all movies played there

```
SELECT S.Theater, AVG(M.Length) AS Average_Length
FROM Schedule S
JOIN Movie M USING (Title)
    GROUP BY S.Theater
    HAVING MAX(M.Length) > 120
```

Exercises 1/2

1. Definitions

SQL, Bag-RA, Nested query, θ-join, Outer join

2. True or False?

i) Every RA query is expressible within SQL.
ii) `SELECT R.A FROM R` $\equiv \pi_A(R)$.
iii) $\{(1),(1),(2)\}$ `EXCEPT` $\{(2),(3)\} = \{(1),(1)\}$.
iv) $(R \cup S) - T = (R - T) \cup (S - T)$ with SQL bag semantics.
v) Boolean expression with a `NULL` value returns True or False.

Exercises 2/2

3. SQL queries ✐

1. Find actors who did not play in a movie by **Chabrol**.
2. Find theaters that play movies not played anywhere else.
3. Write division in the *translated RA* flavor.

4. Problem

"*For each theater playing at least one long (over 2 hours) movie, find the average length of all movies played there.*"
The above aggregate query has a straightforward SQL translation with `GROUP BY/HAVING` statement. Show another way to write SQL query without `GROUP BY/HAVING`. *Hint*: use subqueries in the `SELECT` clause and in the `WHERE` clause.

Database constraints

- In our running examples we assumed that the title attribute identifies a movie
- But this may not be the case:

title	director	length	release_date
Dracula	Browning	84mn	1931
Dracula	Fischer	82mn	1958
Dracula	Badham	109mn	1979
Dracula	Coppola	127mn	1992

- Database constraints: provide additional semantic information about the data
- Most common ones: **functional** and **inclusion** dependencies, and their special cases: keys and foreign keys

Functional dependency

- If we want the title to identify a movie uniquely (i.e., no multiple Dracula records), we express it as a **functional dependency**

$$\text{title} \;\longrightarrow\; \text{director},\; \text{length},\; \text{release\_date}$$

- More generally:

$$X \longrightarrow Y ::= \Big(\forall t, u \in R,\;\; t[X] = u[X] \implies t[Y] = u[Y]\Big)$$

Running Example

R

A	B	C	D	E
a	a	b	a	d
a	b	b	a	d
a	c	c	d	d
b	a	a	a	d
b	b	a	a	d
b	d	d	c	a

Functional dependencies that hold in R are:

$$\mathcal{F} = \{AB \longrightarrow CD, C \longrightarrow ADE, B \longrightarrow DE, D \longrightarrow E\}$$

Keys

Let K be a subset of attributes of $R{:}\langle\vec{U}\rangle$. Then K is a **key** if R satisfies functional dependency $K \longrightarrow U$ and K is minimal

Follow-on of the example

Keys are $\{AB, BC\}$

Among **candidate keys**, one can be arbitrarily promoted into **primary key**

Inclusion constraints

Referential integrity

Attributes of one relation refer to values in another one

- These particular constraints are called **inclusion dependencies** (ID)
- Formally, we have an inclusion dependency $R[X] \subseteq S[Y]$ when every value of the set of attributes X in R also occurs as a value of the set of attributes Y in S:

$$\pi_{\vec{X}}(R) \subseteq \pi_{\vec{Y}}(S)$$

Foreign keys

- Most often IDs occur as part of a **foreign key**
- Foreign key is a conjunction of a key and an ID:

$$R[X] \subseteq S[Y] \text{ and } Y \longrightarrow U, \text{ with } S{:}\langle \vec{U} \rangle$$

Example

We expect `Theater` and `Title` from `Schedule` to be found resp. in `Location` and `Movie`:

- `Schedule[Theater]` $\subseteq$ `Location[Theater]`
- `Schedule[Title]` $\subseteq$ `Movie[Title]`

If `Title` is a key for `Movie`, then it is **foreign key** in `Schedule`
Same arises for `Theater`

Reasoning about FDs

Closure

Denote by $\mathcal{F}$ the set of functional dependancies on R; **Closure** of $\mathcal{F}$ is $\mathcal{F}^+ = \{f \mid \mathcal{F} \models f\}$

Attribute closure of X on $\mathcal{F}$: $X^+_{\mathcal{F}} = \{A \mid X \longrightarrow A \in \mathcal{F}^+\}$

Armstrong's axioms

- **Reflexivity**: if $X \supseteq Y$, then $X \longrightarrow Y$
- **Augmentation**: if $X \longrightarrow Y$, then $XZ \longrightarrow YZ$ for any Z
- **Transitivity**: if $X \longrightarrow Y$ and $Y \longrightarrow Z$, then $X \longrightarrow Z$

- These are sound and complete inference rules for FDs!

Reasoning about FDs (cont'd)

Commonly derived rules:

- **Union**: if $X \longrightarrow Y$ and $X \longrightarrow Z$, then $X \longrightarrow YZ$
- **Decomposition**: if $X \longrightarrow YZ$, then $X \longrightarrow Y$ and $X \longrightarrow Z$
- **Pseudo-transitivity**: if $X \longrightarrow Y$ and $YZ \longrightarrow T$, then $XZ \longrightarrow T$

Back to the example

$$\begin{aligned}\mathcal{F}^+ = &\mathcal{F} \cup \{A \longrightarrow A, AB \longrightarrow A, BC \longrightarrow D,\\ &BC \longrightarrow BC, DA \longrightarrow E, AB \longrightarrow DE, ABCD \longrightarrow A,\\ &ABCD \longrightarrow ABCD, \ldots\}\end{aligned}$$

Canonical cover

On the other side

Canonical cover $\mathcal{F}_{\text{min}}$ such that $\mathcal{F}_{\text{min}}^{+} = \mathcal{F}^{+}$ and FDs in $\mathcal{F}_{\text{min}}$ are all **irreducible**

- Non unique
- Preferred form for normalization

Example

$$\mathcal{F}_{\text{min}} = \{AB \longrightarrow C, C \longrightarrow D, C \longrightarrow A, B \longrightarrow D, D \longrightarrow E\}$$

Idea: Is $AB \longrightarrow D$ redundant in $\mathcal{F}$?
Check for $D \in AB_{\mathcal{G}}^{+}$, $\mathcal{G} = \mathcal{F} - \{AB \longrightarrow D\}$

Database design

Normalization

Avoid redundancy and modification anomalies

Example

Assume AC, BD and DE are three distinct entities in R

1. d value **occurs 5 times** in column E of R
2. **Updating** $E{:}a$ to $E{:}a'$ in the first row requires to update 4 more rows, making the database inconsistent otherwise
3. **Inserting** $\langle A{:}b, C{:}b\rangle$ in R requires to provide BDE values as well
4. **Deleting** last pair $\langle D{:}a, E{:}d\rangle$ in R implies to delete ABC values as well

Normal forms

1NF

Table has (a) a **key** and (b) **atomic** columns and (c) **no repeating groups** of columns

Examples

- R is 1NF
- Sets or tuples or tables are not allowed as attribute values
- (Author_1, Author_2) is not allowed as a subset of columns

Normal forms (cont'd)

2NF

1NF + **full** FD from keys to non-prime attributes

Prime attributes are those that belong to any candidate key

Examples

- R is not 2NF since $B \longrightarrow D$ holds in R and AB is a key and D is a non-prime attribute
- $R_1 = \pi_{BDE}(R)$ and $R_2 = \pi_{ABC}(R)$ are both 2NF

Normal forms (cont'd)

3NF

2NF + **non-transitive** FD from keys to non-prime attributes

Examples

- R_2:$\langle ABC \rangle$ is 3NF with key AB
- R_1:$\langle BDE \rangle$ is not since $D \longrightarrow E$ holds
- $R_{11} = \pi_{BD}(R_1)$ and $R_{12} = \pi_{DE}(R_1)$ are both 2NF

Normal forms (cont'd)

BCNF

3NF + every non-trivial FD is on a **superkey**

Examples

- R_{11}:$\langle BD \rangle$ and R_{12}:$\langle DE \rangle$ are both BCNF
- R_2:$\langle ABC \rangle$ is not since $C \longrightarrow A$ holds
- $R_{21} = \pi_{AC}(R_2)$ and $R_{22} = \pi_{BC}(R_2)$ are both BCNF

Decomposing a relation

Decomposition of R into $S{:}\langle\vec{X}\rangle$ and $T{:}\langle\vec{Y}\rangle$

- Lossless-join decomposition of R: $\pi_{\vec{X}}(R) \bowtie \pi_{\vec{Y}}(R) = R$
- Lossless w.r.t. $\mathcal{F}$:

$$X \cap Y \longrightarrow X, \quad \text{or}$$
$$X \cap Y \longrightarrow Y$$

- Dependency preservation: $(\mathcal{F}_X \cup \mathcal{F}_Y)^+ = \mathcal{F}^+$

Theorem

Lossless-join dependency preserving decomposition of R into a collection of 3NF relations is always possible

What about decomposition into BCNF?

Multivalued and join dependencies

More stringent normal forms

- MVD: 4NF...
- FD+JD: ETNF (H. Darwen et al., 2012)...
- JD: 5NF...
- JD: 6NF...
- Domain and Key constraints: DKNF...

Exercises 1/2

1. Definitions

Functional dependency, Inclusion dependency, Key, Foreign key, Closure, Armstrong's axioms, Canonical cover, Modification anomaly, 1NF, 2NF, 3NF, BCNF

2. True or False?

i) A relation may have several keys.
ii) Closure is not unique.
iii) $\mathcal{F}_{\mathsf{min}} \subseteq \mathcal{F} \subseteq \mathcal{F}^{+}$.
iv) $\emptyset \longrightarrow X$ always holds.
v) $2^n \leq |\mathcal{F}^{+}| \leq 2^{2n}$, where $n = \mathrm{ary}(R)$.

Exercises 2/2

3. Misc.

Consider a database with a single relation $R{:}\langle abcde\rangle$; the set of functional dependencies that hold in R is
$\mathcal{F} = \{ab \rightarrow cd, a \rightarrow d, e \rightarrow b, cd \rightarrow ce, ac \rightarrow bde, c \rightarrow a\}$.

1. Give (i) keys of R, (ii) NF of R, (iii) a canonical cover of $\mathcal{F}$.
2. Decompose R up to the BCNF.

4. Problem — Armstrong Relations

- Prove that there exists an instance of $R{:}\langle\vec{U}\rangle$ such that for each FD f over U, $R \models f$ iff $f \in \mathcal{F}^+$. It is called an **Armstrong relation**.
- Exhibit a set $\mathcal{F}_2$ of FDs over $\{A, B, C\}$ such that each Armstrong relation for $\mathcal{F}_2$ has at least 4 distinct values occurring in the A column.

To Sum-up

1. The relational model is a set-theoretic approach to database
2. Theoretical languages: RA (procedural) $\equiv$ RC (declarative)
3. Two flavors of RA: named (SPJRUD) or unnamed (SPCUD)
4. Nice fragments of RA: SPJR $\equiv$ Conjunctive Queries
5. Practical languages: QBE (Tableau-like queries) and SQL (TRC-like queries at the heart)
6. Many sophisticated statements and extra-features in SQL
7. Functional and incl. dependencies between columns and tables
8. DB design prevents from redundancy and modification anomalies
9. Losseless-join dependency preserving decomposition up to 3NF

Database Management Systems

Main topics are

- **Physical model**: storage, access methods
- **Query optimization**: execution plan, cost evaluation, join algorithms, external sort
- **Transactions and concurrency control**: ACID properties, serializability, 2PL
- **Failure recovery**: logs
- **Tuning**: de-normalization, query tricks, administration

Great opportunities for self-learning!