Query Processing for Non-traditional Applications

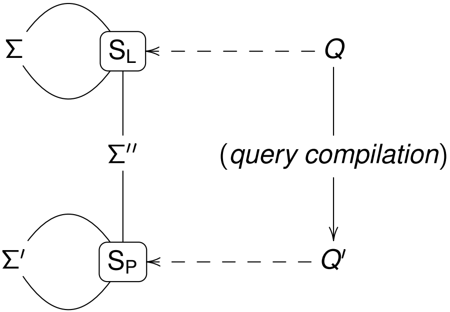
CS848 Spring 2013

Cheriton School of CS

Logical and Physical Schemas

Physical Design and Query Compilation: Overview

$$\Sigma = (\Sigma \cup \Sigma' \cup \Sigma'')$$



Standard Design: Discussion

Standard (relational) Physical Design

`CREATE TABLE employee` DDL command causes

1. a *logical symbol* `employee` to be created;
2. a (disk-based) *file* of (appropriate) records to be created; and
3. a link between these two objects to be *recorded* (where?)

- Multiple indices for the same table
- Horizontal partitioning
- Views and Materialized views

Points to consider

1. How are the above options *recorded* in a RDBMs? (speculation is ok)
2. Is there a uniform and compact way to describe *all* of the above options?

Language(s) for Metadata (et al.)

First-order Logic

1. First-order signatures for S_L and (most of) S_P,
2. First-order sentences for Σ (Σ', Σ''),
3. First order formulae for Q and (most of) Q'.

Language(s) for Metadata (et al.)

First-order Logic

1. First-order signatures for S_L and (most of) S_P,
2. First-order sentences for Σ (Σ', Σ''),
3. First order formulae for Q and (most of) Q'.

⇒ we'll need some additional *auxiliary* information in the cases of

S_P: which attributes are *input parameters*?
what are the symbol's *performance characteristics*?

Q': how are formulae *mapped* to imperative programs?
are there plan features *not* captured by formula syntax?

LOGICAL DESIGN

Syntax of FOL: Signatures

Vocabularies are called *signatures* in FOL.

Syntax of FOL: Signatures

Vocabularies are called *signatures* in FOL.

The *non-logical parameters* in FOL consist of infinite disjoint collections $\{P_1, P_2, \ldots\}$ and $\{f_1, f_2, \ldots\}$ of *predicate symbols* and *function symbols*, respectively.

The *arity* of each symbol is a non-negative integer n, denoted $\mathrm{Ar}(P_i)$ or $\mathrm{Ar}(f_i)$.

- P/i denotes predicate symbol P where $\mathrm{Ar}(P) = i$.
- f/j denotes function symbol f where $\mathrm{Ar}(f) = j$.

Syntax of FOL: Signatures

Vocabularies are called *signatures* in FOL.

The *non-logical parameters* in FOL consist of infinite disjoint collections $\{P_1, P_2, \ldots\}$ and $\{f_1, f_2, \ldots\}$ of *predicate symbols* and *function symbols*, respectively.

The *arity* of each symbol is a non-negative integer n, denoted $\text{Ar}(P_i)$ or $\text{Ar}(f_i)$.

- P/i denotes predicate symbol P where $\text{Ar}(P) = i$.
- f/j denotes function symbol f where $\text{Ar}(f) = j$.

Predicate symbols of arity 0 are also called *propositions*,
Function symbols of arity 0 are also called *constants*.

Syntax of FOL: Signatures

Vocabularies are called *signatures* in FOL.

The *non-logical parameters* in FOL consist of infinite disjoint collections $\{P_1, P_2, \ldots\}$ and $\{f_1, f_2, \ldots\}$ of *predicate symbols* and *function symbols*, respectively.

The *arity* of each symbol is a non-negative integer n, denoted $\text{Ar}(P_i)$ or $\text{Ar}(f_i)$.

- P/i denotes predicate symbol P where $\text{Ar}(P) = i$.
- f/j denotes function symbol f where $\text{Ar}(f) = j$.

Predicate symbols of arity 0 are also called *propositions*,
Function symbols of arity 0 are also called *constants*.

A *signature* S in FOL is a (possibly infinite) selection of non-logical parameters.

- S^P denotes all predicate symbols in S.
- S^F denotes all function symbols in S.

ACME Case: Signatures for PAYROLL

OPTION 1

- $S_L^P = \{\texttt{employee}/3\}$
- $S_L^F = \emptyset$

ACME Case: Signatures for PAYROLL

OPTION 1

- $S_L^P = \{\texttt{employee}/3\}$
- $S_L^F = \emptyset$

Fewest non-logical parameters: a single 3-ary predicate symbol.

- 1st arg: an employee number
- 2nd arg: an employee name
- 3rd arg: an employee salary

ACME Case: Signatures for PAYROLL

OPTION 1

- $\mathsf{S}_{\mathsf{L}}^{\mathsf{P}} = \{\mathtt{employee}/3\}$
- $\mathsf{S}_{\mathsf{L}}^{\mathsf{F}} = \emptyset$

Fewest non-logical parameters: a single 3-ary predicate symbol.

- 1st arg: an employee number
- 2nd arg: an employee name
- 3rd arg: an employee salary

1st arg serves a special role: the set of employee number values is identified with the set of employees.

ACME Case: Signatures for PAYROLL

OPTION 1

- $S_L^P = \{\texttt{employee}/3\}$
- $S_L^F = \emptyset$

Fewest non-logical parameters: a single 3-ary predicate symbol.

- 1st arg: an employee number
- 2nd arg: an employee name
- 3rd arg: an employee salary

1st arg serves a special role: the set of employee number values is identified with the set of employees.

Each 3-tuple in $(\texttt{employee})^{\mathcal{I}}$ suggests two things.

1. The employee number is a *visible object identifier* of some employee.
2. The remaining two components of the 3-tuple express two facts about the employee.

Signatures for PAYROLL

OPTION 2

- $S_L^P = \{\texttt{employee}/1\}$
- $S_L^F = \{\texttt{employee-number}/1, \texttt{name}/1, \texttt{salary}/1\}$

Signatures for PAYROLL

OPTION 2

- $S_L^P = \{\texttt{employee}/1\}$
- $S_L^F = \{\texttt{employee-number}/1, \texttt{name}/1, \texttt{salary}/1\}$

Trades the need to remember the role of argument positions with the need to learn and remember additional non-logical parameters.

Signatures for PAYROLL

OPTION 2

- $S_L^P = \{\texttt{employee}/1\}$
- $S_L^F = \{\texttt{employee-number}/1, \texttt{name}/1, \texttt{salary}/1\}$

Trades the need to remember the role of argument positions with the need to learn and remember additional non-logical parameters.

Introduce

- unary predicates to capture the various kinds of entities, and
- unary functions to capture entity attributes.

Signatures for PAYROLL

OPTION 2

- $S_L^P = \{\texttt{employee}/1\}$
- $S_L^F = \{\texttt{employee-number}/1, \texttt{name}/1, \texttt{salary}/1\}$

Trades the need to remember the role of argument positions with the need to learn and remember additional non-logical parameters.

Introduce

- unary predicates to capture the various kinds of entities, and
- unary functions to capture entity attributes.

Advantages:

- Separates entity classification from entity description: an entity *e* in a given interpretation $\mathcal{I}$ is an employee exactly when $e \in (\texttt{employee})^{\mathcal{I}}$.
- All information about entities, such as a `name` or `salary`, is captured by unary functions.

Signatures for PAYROLL

OPTION 1 versus OPTION 2

Latter allows the possibility that more than one employee can have the same *combination* of values for attributes `employee-number`, `name` and `salary`.

Signatures for PAYROLL

OPTION 1 versus OPTION 2

Latter allows the possibility that more than one employee can have the same *combination* of values for attributes `employee-number`, `name` and `salary`.

Replacing an *n*-ary predicate symbol with one unary predicate symbol and *n* unary function symbols is called *reification*.

Signatures for PAYROLL

OPTION 1 versus OPTION 2

Latter allows the possibility that more than one employee can have the same *combination* of values for attributes `employee-number`, `name` and `salary`.

Replacing an *n*-ary predicate symbol with one unary predicate symbol and *n* unary function symbols is called *reification*.

Disadvantages:

- Requires all entities to have a value for all attributes.

Signatures for PAYROLL

OPTION 1 versus OPTION 2

Latter allows the possibility that more than one employee can have the same *combination* of values for attributes `employee-number`, `name` and `salary`.

Replacing an *n*-ary predicate symbol with one unary predicate symbol and *n* unary function symbols is called *reification*.

Disadvantages:

- Requires all entities to have a value for all attributes.
- Therefore requires simulating partial functions (e.g., “null inapplicable” values).

Signatures for PAYROLL

OPTION 3

- $S_L^P = \{\texttt{employee}/1, \texttt{employee-number}/2, \texttt{name}/2, \texttt{salary}/2\}$
- $S_L^F = \emptyset$

Overcomes disadvantages of OPTION 2: replaces each unary function symbol with a new binary predicate symbol.

Signatures for PAYROLL

OPTION 3

- $S_L^P = \{\texttt{employee}/1, \texttt{employee-number}/2, \texttt{name}/2, \texttt{salary}/2\}$
- $S_L^F = \emptyset$

Overcomes disadvantages of OPTION 2: replaces each unary function symbol with a new binary predicate symbol.

Makes it possible for an entity (including employees) to have any number of employee numbers, names or salaries, including none.

Signatures for PAYROLL

OPTION 3

- $S_L^P = \{\texttt{employee}/1, \texttt{employee-number}/2, \texttt{name}/2, \texttt{salary}/2\}$
- $S_L^F = \emptyset$

Overcomes disadvantages of OPTION 2: replaces each unary function symbol with a new binary predicate symbol.

Makes it possible for an entity (including employees) to have any number of employee numbers, names or salaries, including none.

Replacing function symbols with new predicate symbols is always possible when a function free signature is desired.

Variables and Well-Formed Formulae

Denoted V, the *variables* in FOL are a countably infinite collection of symbols

$$\{x_1, x_2, \ldots\}$$

disjoint from the set of non-logical parameters.

Variables and Well-Formed Formulae

Denoted V, the *variables* in FOL are a countably infinite collection of symbols

$$\{x_1, x_2, \ldots\}$$

disjoint from the set of non-logical parameters.

The following grammars define the *terms*, *atoms* and *well formed formulae* induced by S, denoted TERM(S), ATOM(S) and WFF(S), respectively.

- $\text{Term} ::= x \;\; (\text{where } x \in \text{V}) \mid f(\text{Term}_1, \ldots, \text{Term}_n) \;\; (\text{where } f/n \in \text{S}^{\text{F}})$
- $\text{Atom} ::= \text{Term}_1 \approx \text{Term}_2 \mid P(\text{Term}_1, \ldots, \text{Term}_n) \;\; (\text{where } P/n \in \text{S}^{\text{P}})$
- $\phi, \psi ::= \text{Atom} \mid \neg\,\phi \mid (\phi \wedge \psi) \mid \exists\, x\,.\,\phi \;\; (\text{where } x \in \text{V})$

Variables and Well-Formed Formulae

Denoted V, the *variables* in FOL are a countably infinite collection of symbols

$$\{x_1, x_2, \ldots\}$$

disjoint from the set of non-logical parameters.

The following grammars define the *terms*, *atoms* and *well formed formulae* induced by S, denoted TERM(S), ATOM(S) and WFF(S), respectively.

- Term $::=$ x (where $x \in \mathrm{V}$) $\mid$ $f(\mathrm{Term}_1, \ldots, \mathrm{Term}_n)$ (where $f/n \in \mathrm{S}^{\mathrm{F}}$)
- Atom $::=$ $\mathrm{Term}_1 \approx \mathrm{Term}_2$ $\mid$ $P(\mathrm{Term}_1, \ldots, \mathrm{Term}_n)$ (where $P/n \in \mathrm{S}^{\mathrm{P}}$)
- ϕ, ψ $::=$ Atom $\mid$ $\neg\,\phi$ $\mid$ $(\phi \wedge \psi)$ $\mid$ $\exists\, x \,.\, \phi$ (where $x \in \mathrm{V}$)

Assumes S denotes an FOL signature; we omit S when clear from context.

Variables and Well-Formed Formulae (cont'd)

The *logical parameters* in FOL:

- (*equality*) $\approx$
- (*negation*) $\neg$
- (*conjunction*) $\wedge$
- (*existential quantification*) $\exists$

Variables and Well-Formed Formulae (cont'd)

The *logical parameters* in FOL:

- (*equality*) $\approx$
- (*negation*) $\neg$
- (*conjunction*) $\wedge$
- (*existential quantification*) $\exists$

Convenient to have additional logical parameters as syntactic shorthand:

- (*disjunction*) $\vee$: "$(\phi \vee \psi)$" $\leadsto$ "$\neg(\neg\phi \wedge \neg\psi)$"
- (*implication*) $\rightarrow$: "$(\phi \rightarrow \psi)$" $\leadsto$ "$(\neg\phi \vee \psi)$"
- (*equivalence*) $\equiv$: "$(\phi \equiv \psi)$" $\leadsto$ "$((\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi))$"
- (*universal quantification*) $\forall$: "$\forall x.\phi$" $\leadsto$ "$\neg\exists x.\neg\phi$".

Variables and Well-Formed Formulae (cont'd)

The *logical parameters* in FOL:

- (*equality*) $\approx$
- (*negation*) $\neg$
- (*conjunction*) $\wedge$
- (*existential quantification*) $\exists$

Convenient to have additional logical parameters as syntactic shorthand:

- (*disjunction*) $\vee$: "$(\phi \vee \psi)$" $\rightsquigarrow$ "$\neg(\neg\phi \wedge \neg\psi)$"
- (*implication*) $\rightarrow$: "$(\phi \rightarrow \psi)$" $\rightsquigarrow$ "$(\neg\phi \vee \psi)$"
- (*equivalence*) $\equiv$: "$(\phi \equiv \psi)$" $\rightsquigarrow$ "$((\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi))$"
- (*universal quantification*) $\forall$: "$\forall x.\phi$" $\rightsquigarrow$ "$\neg\exists x.\neg\phi$".

Also common practice to omit parenthesis in well-formed formulae when intentions are clear, e.g.:

"$(\phi_1 \wedge \phi_2 \wedge \phi_3)$" instead of "$(\phi_1 \wedge (\phi_2 \wedge \phi_3))$"

Free Variables

Given $t \in$ TERM or $\phi \in$ WFF: $\mathrm{Fv}(t)$ and $\mathrm{Fv}(\phi)$ denote the *free variables* of a term and of a well formed formula, respectively.

$$\mathrm{Fv}(t) = \begin{cases} \{x\} & \text{if } t = \text{“}x\text{”, and} \\ \bigcup_{1 \le i \le n} \mathrm{Fv}(t_i) & \text{when } t = \text{“}f(t_1, \ldots, t_n)\text{” otherwise.} \end{cases}$$

$$\mathrm{Fv}(\phi) = \begin{cases} \bigcup_{1 \le i \le n} \mathrm{Fv}(t_i) & \text{if } \phi = \text{“}P(t_1, \ldots, t_n)\text{”,} \\ \mathrm{Fv}(t_1) \cup \mathrm{Fv}(t_2) & \text{if } \phi = \text{“}t_1 \approx t_2\text{”,} \\ \mathrm{Fv}(\psi) & \text{if } \phi = \text{“}\neg\, \psi\text{”,} \\ \mathrm{Fv}(\psi_1) \cup \mathrm{Fv}(\psi_2) & \text{if } \phi = \text{“}(\psi_1 \wedge \psi_2)\text{”, and} \\ \mathrm{Fv}(\psi) - \{x\} & \text{when } \phi = \text{“}\exists x.\psi\text{” otherwise.} \end{cases}$$

A well-formed formula ϕ is *closed* if $\mathrm{Fv}(\phi) = \emptyset$. A closed well-formed formula is also called a *sentence*.

Semantics of FOL: Interpretations

Assume S denotes a signature in FOL.
An *interpretation* $\mathcal{I}(\mathsf{S})$ of S is a pair $\langle \triangle^{\mathcal{I}(\mathsf{S})}, (\cdot)^{\mathcal{I}(\mathsf{S})} \rangle$.

1. $\triangle^{\mathcal{I}(\mathsf{S})}$ is a non-empty *domain* of entities.
2. $(\cdot)^{\mathcal{I}(\mathsf{S})}$ is an *interpretation function*.

Semantics of FOL: Interpretations

Assume S denotes a signature in FOL.
An *interpretation* $\mathcal{I}(\mathsf{S})$ of S is a pair $\langle \triangle^{\mathcal{I}(\mathsf{S})}, (\cdot)^{\mathcal{I}(\mathsf{S})} \rangle$.

1. $\triangle^{\mathcal{I}(\mathsf{S})}$ is a non-empty *domain* of entities.
2. $(\cdot)^{\mathcal{I}(\mathsf{S})}$ is an *interpretation function*.

For each $P/n \in \mathsf{S}_\mathsf{P}$, $(P/n)^{\mathcal{I}(\mathsf{S})}$ is a subset of $(\triangle^{\mathcal{I}(\mathsf{S})})^n$.

Semantics of FOL: Interpretations

Assume S denotes a signature in FOL.
An *interpretation* $\mathcal{I}(\mathsf{S})$ of S is a pair $\langle \triangle^{\mathcal{I}(\mathsf{S})}, (\cdot)^{\mathcal{I}(\mathsf{S})} \rangle$.

1. $\triangle^{\mathcal{I}(\mathsf{S})}$ is a non-empty *domain* of entities.
2. $(\cdot)^{\mathcal{I}(\mathsf{S})}$ is an *interpretation function*.

For each $P/n \in \mathsf{S_P}$, $(P/n)^{\mathcal{I}(\mathsf{S})}$ is a subset of $(\triangle^{\mathcal{I}(\mathsf{S})})^n$.

For each $f/n \in \mathsf{S_F}$, $(f/n)^{\mathcal{I}(\mathsf{S})}$ is a total function: $(\triangle^{\mathcal{I}(\mathsf{S})})^n \rightarrow \triangle^{\mathcal{I}(\mathsf{S})}$.

Semantics of FOL: Interpretations

Assume S denotes a signature in FOL.
An *interpretation* $\mathcal{I}(\mathsf{S})$ of S is a pair $\langle \triangle^{\mathcal{I}(\mathsf{S})}, (\cdot)^{\mathcal{I}(\mathsf{S})} \rangle$.

1. $\triangle^{\mathcal{I}(\mathsf{S})}$ is a non-empty *domain* of entities.
2. $(\cdot)^{\mathcal{I}(\mathsf{S})}$ is an *interpretation function*.

For each $P/n \in \mathsf{S}_\mathsf{P}$, $(P/n)^{\mathcal{I}(\mathsf{S})}$ is a subset of $(\triangle^{\mathcal{I}(\mathsf{S})})^n$.

For each $f/n \in \mathsf{S}_\mathsf{F}$, $(f/n)^{\mathcal{I}(\mathsf{S})}$ is a total function: $(\triangle^{\mathcal{I}(\mathsf{S})})^n \rightarrow \triangle^{\mathcal{I}(\mathsf{S})}$.

Write $\langle e_1, \ldots, e_n \rangle$ to denote an *n-tuple*, an element of $(\triangle^{\mathcal{I}(\mathsf{S})})^n$.

Valuations

Assume $\mathcal{I}$ is an interpretation of signature S.

Valuations

Assume $\mathcal{I}$ is an interpretation of signature S.

A *valuation* over $\mathcal{I}$ is written $\mathcal{V}(\mathcal{I})$ (or as $\mathcal{V}$ when $\mathcal{I}$ is clear from context) and is a total function: V $\rightarrow \triangle^{\mathcal{I}}$.

Valuations

Assume $\mathcal{I}$ is an interpretation of signature S.

A *valuation* over $\mathcal{I}$ is written $\mathcal{V}(\mathcal{I})$ (or as $\mathcal{V}$ when $\mathcal{I}$ is clear from context) and is a total function: $\mathsf{V} \to \triangle^{\mathcal{I}}$.

For a given $x \in \mathsf{V}$ and $e \in \triangle^{\mathcal{I}}$, the valuation $\mathcal{V}[x \mapsto e]$ is defined as follows:

$$\mathcal{V}[x_1 \mapsto e](x_2) = \begin{cases} e & \text{if "}x_1\text{" = "}x_2\text{", and} \\ \mathcal{V}(x_2) & \text{otherwise.} \end{cases}$$

Valuations

Assume $\mathcal{I}$ is an interpretation of signature S.

A *valuation* over $\mathcal{I}$ is written $\mathcal{V}(\mathcal{I})$ (or as $\mathcal{V}$ when $\mathcal{I}$ is clear from context) and is a total function: $\mathrm{V} \to \triangle^{\mathcal{I}}$.

For a given $x \in \mathrm{V}$ and $e \in \triangle^{\mathcal{I}}$, the valuation $\mathcal{V}[x \mapsto e]$ is defined as follows:

$$\mathcal{V}[x_1 \mapsto e](x_2) = \begin{cases} e & \text{if "}x_1\text{"} = \text{"}x_2\text{", and} \\ \mathcal{V}(x_2) & \text{otherwise.} \end{cases}$$

A valuation $\mathcal{V}$ is extended to apply to any $t \in \mathrm{TERM}$ in *the* way that satisfies

$$\mathcal{V}(t) = (f)^{\mathcal{I}}(\mathcal{V}(t_1), \ldots, \mathcal{V}(t_n))$$

whenever $t =$ "$f(t_1, \ldots, t_n)$".

Models

Assume S is a signature in FOL and also that $\phi \in$ WFF(S).

Models

Assume S is a signature in FOL and also that $\phi \in$ WFF(S).

An interpretation $\mathcal{I}$ of S and valuation $\mathcal{V}$ over $\mathcal{I}$ is a *model* of ϕ, written

$$\mathcal{I}, \mathcal{V} \models \phi,$$

iff one of the following conditions apply:

- $\phi =$ "$P(t_1, \ldots, t_n)$" and $\langle \mathcal{V}(t_1), \ldots, \mathcal{V}(t_n) \rangle \in (P)^{\mathcal{I}}$,
- $\phi =$ "$t_1 \approx t_2$" and $\mathcal{V}(t_1) = \mathcal{V}(t_2)$,
- $\phi =$ "$\neg\psi$" and $\mathcal{I}, \mathcal{V} \not\models \psi$,
- $\phi =$ "$(\psi_1 \wedge \psi_2)$", $\mathcal{I}, \mathcal{V} \models \psi_1$ and $\mathcal{I}, \mathcal{V} \models \psi_2$, or
- $\phi =$ "$\exists x.\psi$" and $\mathcal{I}, \mathcal{V}[x \mapsto e] \models \psi$ for some $e \in \triangle^{\mathcal{I}}$.

Satisfiability and Logical Consequence

Assume Σ is a theory (over signature S).

Satisfiability and Logical Consequence

Assume Σ is a theory (over signature S).

We say the following.

1. The pair $\mathcal{I}, \mathcal{V}$ is a *model* of Σ if $\mathcal{I}, \mathcal{V} \models \psi$ for all $\psi \in \Sigma$.

Satisfiability and Logical Consequence

Assume Σ is a theory (over signature S).

We say the following.

1. The pair $\mathcal{I}, \mathcal{V}$ is a *model* of Σ if $\mathcal{I}, \mathcal{V} \models \psi$ for all $\psi \in \Sigma$.
2. Σ is *satisfiable* if it has a model and *unsatisfiable* otherwise.

Satisfiability and Logical Consequence

Assume Σ is a theory (over signature S).

We say the following.

1. The pair $\mathcal{I}, \mathcal{V}$ is a *model* of Σ if $\mathcal{I}, \mathcal{V} \models \psi$ for all $\psi \in \Sigma$.
2. Σ is *satisfiable* if it has a model and *unsatisfiable* otherwise.
3. ϕ is a *logical consequence* of Σ, written

$$\Sigma \models \phi,$$

   iff $\mathcal{I}, \mathcal{V} \models \phi$ for any model $\mathcal{I}, \mathcal{V}$ of Σ.

Satisfiability and Logical Consequence

Assume Σ is a theory (over signature S).

We say the following.

1. The pair $\mathcal{I}, \mathcal{V}$ is a *model* of Σ if $\mathcal{I}, \mathcal{V} \models \psi$ for all $\psi \in \Sigma$.
2. Σ is *satisfiable* if it has a model and *unsatisfiable* otherwise.
3. ϕ is a *logical consequence* of Σ, written

$$\Sigma \models \phi,$$

iff $\mathcal{I}, \mathcal{V} \models \phi$ for any model $\mathcal{I}, \mathcal{V}$ of Σ.

The fundamental problem of reasoning in a given FOL theory Σ(S) is the problem of *logical implication*: establishing which $\phi \in$ WFF(S) are logical consequences of Σ(S).

ACME Case: Logical Constraints for PAYROLL

On identification.

Assume S_L is given by OPTION 1.

ACME Case: Logical Constraints for PAYROLL

On identification.

Assume S_L is given by OPTION 1.

The condition that *employees can be identified by their employee number* can be expressed as the FOL sentence

$$\forall x_1, x_2, y_1, y_2.(\exists z.(\texttt{employee}(z, x_1, x_2) \land \texttt{employee}(z, y_1, y_2)) \\ \to ((x_1 \approx y_1) \land (x_2 \approx y_2))).$$

ACME Case: Logical Constraints for PAYROLL

On identification.

Assume S_L is given by OPTION 1.

The condition that *employees can be identified by their employee number* can be expressed as the FOL sentence

$$\forall x_1, x_2, y_1, y_2.(\exists z.(\texttt{employee}(z, x_1, x_2) \wedge \texttt{employee}(z, y_1, y_2)) \\ \rightarrow ((x_1 \approx y_1) \wedge (x_2 \approx y_2))).$$

Ensures that each employee is associated with a single 3-tuple in $(\texttt{employee})^{\mathcal{I}}$ in any interpretation $\mathcal{I}$ for ACME's PAYROLL system.

ACME Case: Logical Constraints for PAYROLL

On identification.

Assume S_L is given by OPTION 1.

The condition that *employees can be identified by their employee number* can be expressed as the FOL sentence

$$\forall x_1, x_2, y_1, y_2.(\exists z.(\texttt{employee}(z, x_1, x_2) \wedge \texttt{employee}(z, y_1, y_2)) \\ \rightarrow ((x_1 \approx y_1) \wedge (x_2 \approx y_2))).$$

Ensures that each employee is associated with a single 3-tuple in $(\texttt{employee})^{\mathcal{I}}$ in any interpretation $\mathcal{I}$ for ACME's PAYROLL system.[1]

[1] Remember introductory comments: the collection of all data corresponds to an interpretation $\mathcal{I}$.

ACME Case: Logical Constraints for PAYROLL

On identification.

Assume S_L is given by OPTION 1.

The condition that *employees can be identified by their employee number* can be expressed as the FOL sentence

$$\forall x_1, x_2, y_1, y_2.(\exists z.(\texttt{employee}(z, x_1, x_2) \wedge \texttt{employee}(z, y_1, y_2)) \rightarrow ((x_1 \approx y_1) \wedge (x_2 \approx y_2))).$$

Ensures that each employee is associated with a single 3-tuple in $(\texttt{employee})^{\mathcal{I}}$ in any interpretation $\mathcal{I}$ for ACME's PAYROLL system.[1]

Called a *functional dependency* in relational schema.

[1] Remember introductory comments: the collection of all data corresponds to an interpretation $\mathcal{I}$.

Logical Constraints for PAYROLL (identification cont'd)

For S_L given by OPTION 2:

$$\begin{array}{l} \forall x, y.(\\ \quad (\texttt{employee}(x) \land \texttt{employee}(y) \\ \quad\quad \land\, \texttt{employee-number}(x) \approx \texttt{employee-number}(y)) \\ \quad \to x \approx y). \end{array}$$

Logical Constraints for PAYROLL (identification cont'd)

For S_L given by OPTION 2:

$$\forall x, y.((\texttt{employee}(x) \land \texttt{employee}(y) \land \texttt{employee-number}(x) \approx \texttt{employee-number}(y)) \to x \approx y).$$

For S_L given by OPTION 3:

$$\forall x, y.(\exists z.(\texttt{employee}(x) \land \texttt{employee}(y) \land \texttt{employee-number}(x, z) \land \texttt{employee-number}(y, z)) \to x \approx y).$$

Logical Constraints for PAYROLL (identification cont'd)

For S_L given by OPTION 2:

$$\forall x, y.(\\ (\texttt{employee}(x) \land \texttt{employee}(y) \\ \land \texttt{employee-number}(x) \approx \texttt{employee-number}(y)) \\ \to x \approx y).$$

For S_L given by OPTION 3:

$$\forall x, y.(\\ \exists z.(\texttt{employee}(x) \land \texttt{employee}(y) \\ \land \texttt{employee-number}(x, z) \land \texttt{employee-number}(y, z)) \\ \to x \approx y).$$

More accurately, latter states that: *no pair of distinct employees may have any employee number at all in common* (becomes possible in OPTION 3 for employees to have any number of employee numbers).

Logical Constraints for PAYROLL

On property functionality.

Assume OPTION 3 chosen by ACME's APS department.

Logical Constraints for PAYROLL

On property functionality.

Assume OPTION 3 chosen by ACME's APS department.

Then necessary to disallow the number of possible values for an `employee-number`, `name`, or `salary` attribute for a given employee to exceed one.

Must add constraints to the logical constraints Σ to ensure the attributes are *partial functions*.

$$\forall x, y.(\exists z.(\texttt{employee-number}(z, x) \land \texttt{employee-number}(z, y)) \rightarrow (x \approx y))$$

$$\forall x, y.(\exists z.(\texttt{name}(z, x) \land \texttt{name}(z, y)) \rightarrow (x \approx y))$$

$$\forall x, y.(\exists z.(\texttt{salary}(z, x) \land \texttt{salary}(z, y)) \rightarrow (x \approx y))$$

Logical Constraints for PAYROLL

On typing.

The additional unary predicates in the PAYROLL signature can be used to ensure that attribute values are of appropriate types.

Logical Constraints for PAYROLL

On typing.

The additional unary predicates in the PAYROLL signature can be used to ensure that attribute values are of appropriate types.

For S_L given by OPTION 1:

$$\forall x, y, z.(\texttt{employee}(x, y, z) \\ \rightarrow (\texttt{integer}(x) \wedge \texttt{string}(y) \wedge \texttt{integer}(z)))$$

Logical Constraints for PAYROLL

On typing.

The additional unary predicates in the PAYROLL signature can be used to ensure that attribute values are of appropriate types.

For S_L given by OPTION 1:

$$\forall x, y, z.(\texttt{employee}(x, y, z) \\ \rightarrow (\texttt{integer}(x) \wedge \texttt{string}(y) \wedge \texttt{integer}(z)))$$

For S_L given by OPTION 2:

$$\forall x.(\texttt{employee}(x) \rightarrow (\texttt{integer}(\texttt{employee-number}(x)) \\ \wedge \texttt{string}(\texttt{name}(x)) \\ \wedge \texttt{integer}(\texttt{salary}(x))))$$

Logical Constraints for PAYROLL

For S_L given by OPTION 3:

$$\forall x.(\texttt{employee}(x) \to \exists y, z, w.(\texttt{employee-number}(x, y) \land \texttt{integer}(y)$$
$$\land\ \texttt{name}(x, z) \land \texttt{string}(z))$$
$$\land\ \texttt{salary}(x, w) \land \texttt{integer}(w)))$$

Logical Constraints for PAYROLL

For S_L given by OPTION 3:

$$\forall x.(\texttt{employee}(x) \rightarrow \exists y, z, w.(\texttt{employee-number}(x, y) \wedge \texttt{integer}(y)$$
$$\wedge\ \texttt{name}(x, z) \wedge \texttt{string}(z))$$
$$\wedge\ \texttt{salary}(x, w) \wedge \texttt{integer}(w)))$$

OPTION 3 also makes it possible to say that *only employees have employee numbers*.

$$\forall x.(\exists y.\texttt{employee-number}(x, y) \rightarrow \texttt{employee}(x))$$

PHYSICAL DESIGN

(take 1)

Access Paths and Simple Scanning

Assume ACME's DBA department selects a very simple physical design for PAYROLL: all employee information is recorded in a main-memory array.

```
array emp-array [1 to n] of
   integer emp-num
   integer emp-salary
   string  emp-name
```

Access Paths and Simple Scanning

Assume ACME's DBA department selects a very simple physical design for PAYROLL: all employee information is recorded in a main-memory array.

```
array emp-array [1 to n] of
   integer emp-num
   integer emp-salary
   string  emp-name
```

1. The `salary`, `employee-number` and `name` of each `employee` is recorded at some position in the array (in corresponding fields).
2. DBA ensures array entries are ordered by a *major sort* on `emp-num` values.

Access Paths and Simple Scanning

To capture a physical design in FOL, think in terms of *capabilities* attached to new predicate symbols (that become part of a physical vocabulary).

Access Paths and Simple Scanning

To capture a physical design in FOL, think in terms of *capabilities* attached to new predicate symbols (that become part of a physical vocabulary).

The organization of array `emp-array` suggests two capabilities in particular.

Access Paths and Simple Scanning

To capture a physical design in FOL, think in terms of *capabilities* attached to new predicate symbols (that become part of a physical vocabulary).

The organization of array `emp-array` suggests two capabilities in particular.

1. Scanning all entries: `emp-array0`/3.

Access Paths and Simple Scanning

To capture a physical design in FOL, think in terms of *capabilities* attached to new predicate symbols (that become part of a physical vocabulary).

The organization of array `emp-array` suggests two capabilities in particular.

1. Scanning all entries: `emp-array0`/3.
2. Scanning all entries with the first field matching a given `num` value: `emp-array1`/3.

Access Paths and Simple Scanning

To capture a physical design in FOL, think in terms of *capabilities* attached to new predicate symbols (that become part of a physical vocabulary).

The organization of array `emp-array` suggests two capabilities in particular.[1]

1. Scanning all entries: `emp-array0`/3.
2. Scanning all entries with the first field matching a given `num` value: `emp-array1`/3.

---

[1] The DBA department must provide the code that *implements* these capabilities in a library or at runtime, e.g., code that performs a binary search of `emp-array` in the case of `emp-array0` and `emp-array1`.

Access Paths and Simple Scanning

The *access paths* of S are a distinguished subset S_A of the predicate symbols S_P that comprise the physical vocabulary of S.

Access Paths and Simple Scanning

The *access paths* of S are a distinguished subset S_A of the predicate symbols S_P that comprise the physical vocabulary of S.

The *binding pattern* of an access path $P \in S_A$ is denoted $\mathrm{Bp}(P)$ and is a non-negative integer satisfying $0 \leq \mathrm{Bp}(P) \leq \mathrm{Ar}(P)$.

Access Paths and Simple Scanning

The *access paths* of S are a distinguished subset S_A of the predicate symbols S_P that comprise the physical vocabulary of S.

The *binding pattern* of an access path $P \in S_A$ is denoted $\text{Bp}(P)$ and is a non-negative integer satisfying $0 \leq \text{Bp}(P) \leq \text{Ar}(P)$.

Write $P/n/m$ to indicate that P is a predicate symbol with arity n and that P is also an access path with binding pattern m (and write $P/n/m \in S_A$).

Access Paths and Simple Scanning

The *access paths* of S are a distinguished subset S_A of the predicate symbols S_P that comprise the physical vocabulary of S.

The *binding pattern* of an access path $P \in S_A$ is denoted $\mathrm{Bp}(P)$ and is a non-negative integer satisfying $0 \leq \mathrm{Bp}(P) \leq \mathrm{Ar}(P)$.

Write $P/n/m$ to indicate that P is a predicate symbol with arity n and that P is also an access path with binding pattern m (and write $P/n/m \in S_A$).

The declaration of `emp-array` is captured by adding two new predicate symbols that are also access paths to S_A.

$$\{\texttt{emp-array0}/3/0, \texttt{emp-array1}/3/1\}$$

Access Paths and Simple Scanning

The *access paths* of S are a distinguished subset S_A of the predicate symbols S_P that comprise the physical vocabulary of S.

The *binding pattern* of an access path $P \in S_A$ is denoted $\mathrm{Bp}(P)$ and is a non-negative integer satisfying $0 \leq \mathrm{Bp}(P) \leq \mathrm{Ar}(P)$.

Write $P/n/m$ to indicate that P is a predicate symbol with arity n and that P is also an access path with binding pattern m (and write $P/n/m \in S_A$).

The declaration of `emp-array` is captured by adding two new predicate symbols that are also access paths to S_A.

$$\{\texttt{emp-array0}/3/0, \texttt{emp-array1}/3/1\}$$

This new set of predicate symbols is now the *physical vocabulary* S_P of ACME's PAYROLL system.

ACME Case: Mapping Constraints

How do we know (ensure) that `employee` is properly represented by `emp-array` (and in turn by the access paths `emp-array0` and `emp-array1`)?

ACME Case: Mapping Constraints

How do we know (ensure) that `employee` is properly represented by `emp-array` (and in turn by the access paths `emp-array0` and `emp-array1`)?

ACME's DBA group must specify *mapping* or *correspondence* constraints Σ'' over the signature $(S_L \cup S_P)$.

ACME Case: Mapping Constraints

How do we know (ensure) that `employee` is properly represented by `emp-array` (and in turn by the access paths `emp-array0` and `emp-array1`)?

ACME's DBA group must specify *mapping* or *correspondence* constraints Σ'' over the signature $(\mathsf{S_L} \cup \mathsf{S_P})$.

Such constraints provide the necessary "connections" for all possible interpretations $\mathcal{I}$ (encoding factual data) of the *logical vocabulary* $\mathsf{S_L}$ and the *physical vocabulary* $\mathsf{S_P}$ of PAYROLL.

ACME Case: Mapping Constraints

How do we know (ensure) that `employee` is properly represented by `emp-array` (and in turn by the access paths `emp-array0` and `emp-array1`)?

ACME's DBA group must specify *mapping* or *correspondence* constraints Σ'' over the signature $(\mathsf{S_L} \cup \mathsf{S_P})$.

Such constraints provide the necessary "connections" for all possible interpretations $\mathcal{I}$ (encoding factual data) of the *logical vocabulary* $\mathsf{S_L}$ and the *physical vocabulary* $\mathsf{S_P}$ of PAYROLL.

With OPTION 1 for $\mathsf{S_L}$, DBA can add the following sentences to Σ.

1. $\forall x, y, z.(\texttt{employee}(x, y, z) \rightarrow \texttt{emp-array0}(x, z, y))$
2. $\forall x, y, z.(\texttt{emp-array0}(x, z, y) \rightarrow \texttt{employee}(x, y, z))$

ACME Case: Access Path so far

1. `emp-array0/3/0` allows us to *scan* all employees;
2. `emp-array1/3/1` allows us to *find* (all) employees given `enumber`.

ACME Case: Access Path so far

1. `emp-array0/3/0` allows us to *scan* all employees;
2. `emp-array1/3/1` allows us to *find* (all) employees given `enumber`.

What if we also want to find emplouees *by their name*?

1. use `emp-array0` and *filter out* non-matching employees (“selection”)

ACME Case: Access Path so far

1. `emp-array0/3/0` allows us to *scan* all employees;
2. `emp-array1/3/1` allows us to *find* (all) employees given `enumber`.

What if we also want to find emplouees *by their name*?

1. use `emp-array0` and *filter out* non-matching employees (“selection”)
2. *improve the physical design* to allow efficient search ⇒ “`create index`”

ACME Case: Access Path Code Templates

Note: Code templates for access paths must be provided by ACME’s DBA department.

ACME Case: Access Path Code Templates

Note: Code templates for access paths must be provided by ACME's DBA department.

E.g., Pseudo-code templates realizing a `first/next` protocol for `emp-array0` might be given as follows (variables would be renamed for each occurrence of `emp-array0` in a query plan).

```
function emp-array0-first
  i := 0
  return emp-array0-next
```

```
function emp-array0-next
  i := i + 1
  if (i > n) return false
  x1 := emp-array[i].emp-salary
  x2 := emp-array[i].emp-num
  x3 := emp-array[i].emp-name
  return true
```

ACME Case: Access Path Code Templates

Note: Code templates for access paths must be provided by ACME's DBA department.

E.g., Pseudo-code templates realizing a `first/next` protocol for `emp-array0` might be given as follows (variables would be renamed for each occurrence of `emp-array0` in a query plan).

```
function emp-array0-first
  i := 0
  return emp-array0-next
```

```
function emp-array0-next
  i := i + 1
  if (i > n) return false
  x1 := emp-array[i].emp-salary
  x2 := emp-array[i].emp-num
  x3 := emp-array[i].emp-name
  return true
```

Assumes a global state recording bindings of (possible copies of) variables.

1. x_1, x_2 and x_3 to communicate the contents of `emp-array`.
2. i and n to record scanning status and size of `emp-array`.

Access Path Code Templates

Examples of atomic query plans have so far been based on using an array as a *basic collection type*.

Access Path Code Templates

Examples of atomic query plans have so far been based on using an array as a *basic collection type*. Alternatives to an array could also have served the same purpose: linked lists, simple search trees, and so on.

Access Path Code Templates

Examples of atomic query plans have so far been based on using an array as a *basic collection type*. Alternatives to an array could also have served the same purpose: linked lists, simple search trees, and so on.

It is beyond the scope of this book to consider the synthesis of such basic data structures as part of the job of query compilation.

Access Path Code Templates

Examples of atomic query plans have so far been based on using an array as a *basic collection type*. Alternatives to an array could also have served the same purpose: linked lists, simple search trees, and so on.

It is beyond the scope of this book to consider the synthesis of such basic data structures as part of the job of query compilation.

However, we will see how physical design based on more complex data structures can be usefully *decomposed* into such basic data structures using FOL.

Access Path Code Templates

Examples of atomic query plans have so far been based on using an array as a *basic collection type*. Alternatives to an array could also have served the same purpose: linked lists, simple search trees, and so on.

It is beyond the scope of this book to consider the synthesis of such basic data structures as part of the job of query compilation.

However, we will see how physical design based on more complex data structures can be usefully *decomposed* into such basic data structures using FOL.[1]

[1] A decomposition of more complex data structures can enable compilation opportunities that would otherwise not be possible.

Access Path Code Templates

Examples of atomic query plans have so far been based on using an array as a *basic collection type*. Alternatives to an array could also have served the same purpose: linked lists, simple search trees, and so on.

It is beyond the scope of this book to consider the synthesis of such basic data structures as part of the job of query compilation.

However, we will see how physical design based on more complex data structures can be usefully *decomposed* into such basic data structures using FOL.[1]

Main point: Once given `first/next` "black box" code templates for the basic data structures (such as records, arrays, linked lists and simple search trees) constraints can then be expressed in FOL that do the rest.

[1] A decomposition of more complex data structures can enable compilation opportunities that would otherwise not be possible.

Summary: Data vs. Metadata in FOL

Metadata (database schema)

1. Signature S_L and constranints Σ for the *logical schema*,
2. Signature S_P and constranints Σ' for the *physical schema*,
3. Constraints Σ'' that relate S_L to S_P.

Data (database instance)

A *first-order structure (interpretation)* that

1. interprets symbols in S_L and S_P and
2. satisfies $\Sigma \cup \Sigma' \cup \Sigma''$.