Query Processing for Non-traditional Applications

CS848 Spring 2013

Cheriton School of CS

Advanced Physical Designs

Outline

Consider how practical physical design can be accomplished.

Outline

Consider how practical physical design can be accomplished.

One obtains a practical physical design by doing the following.

1. Adding/declaring new predicate symbols and access paths.

Outline

Consider how practical physical design can be accomplished.

One obtains a practical physical design by doing the following.

1. Adding/declaring new predicate symbols and access paths.
2. Adding new constraints.

Outline

Consider how practical physical design can be accomplished.

One obtains a practical physical design by doing the following.

1. Adding/declaring new predicate symbols and access paths.
2. Adding new constraints.

Topics

- References, pointers and linked structures.
- Nulls, partitions and run-time typing.
- Built-in functions and hashing.
- Two-level store.
- Examples relating to ACME's PAYROLL system.

Outline

Consider how practical physical design can be accomplished.

One obtains a practical physical design by doing the following.

1. Adding/declaring new predicate symbols and access paths.
2. Adding new constraints.

Topics

- References, pointers and linked structures.
- Nulls, partitions and run-time typing.
- Built-in functions and hashing.
- Two-level store.
- Examples relating to ACME's PAYROLL system.[1]

---

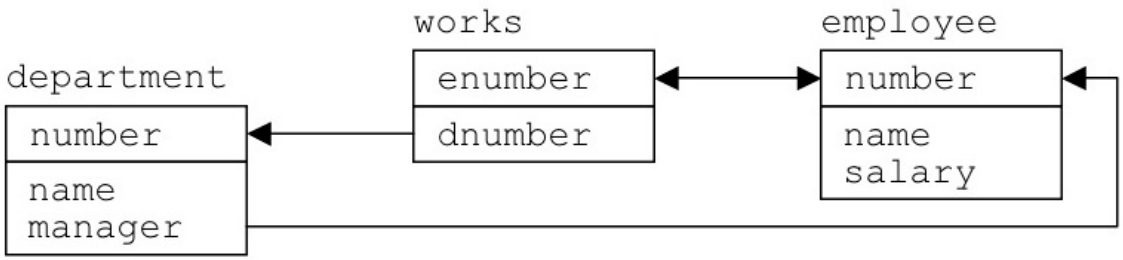
[1] In particular, an extension of the OPTION 1 logical design.

ACME Case: Extending OPTION 1

Assume the APS department extends the logical signature S_L for PAYROLL given by OPTION 1 with two additional predicate symbols to record information about how employees relate to departments.

- $S_L^P = \{\texttt{employee}/3, \texttt{department}/3, \texttt{works}/2\}$, and
- $S_L^F = \emptyset$.

ACME Case: Extending OPTION 1

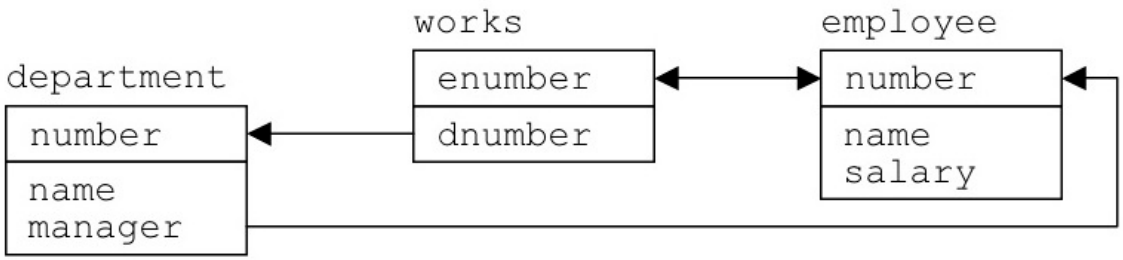


Assume the APS department extends the logical signature S_L for PAYROLL given by OPTION 1 with two additional predicate symbols to record information about how employees relate to departments.

- $S_L^P = \{\texttt{employee}/3, \texttt{department}/3, \texttt{works}/2\}$, and
- $S_L^F = \emptyset$.

Also assume the APS department decides on the logical design illustrated above.

ACME Case: Extending OPTION 1

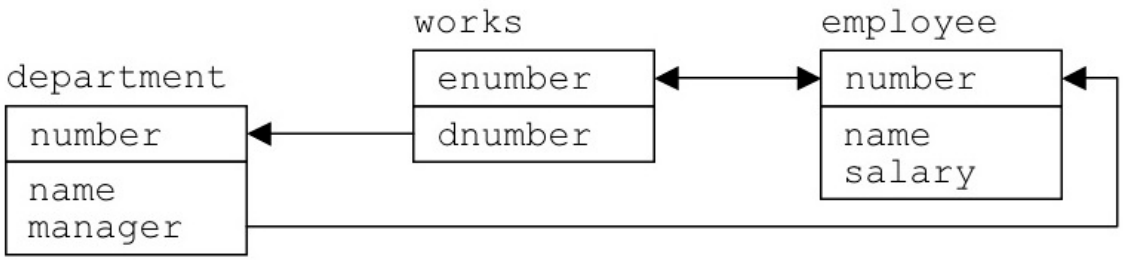


Assume the APS department extends the logical signature S_L for PAYROLL given by OPTION 1 with two additional predicate symbols to record information about how employees relate to departments.

- $S_L^P = \{\texttt{employee}/3, \texttt{department}/3, \texttt{works}/2\}$, and
- $S_L^F = \emptyset$.

Also assume the APS department decides on the logical design illustrated above. Note: Arguments are given *attribute names*; attributes that are *primary keys* are listed first; arrows correspond to *foreign keys*.

ACME Case: Extended OPTION 1

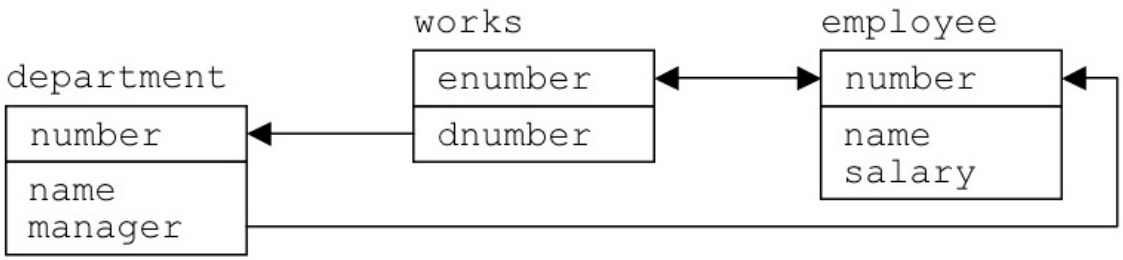


Logical Constraints Σ

1. *Employees can be identified by their number.*

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

ACME Case: Extended OPTION 1



Logical Constraints Σ

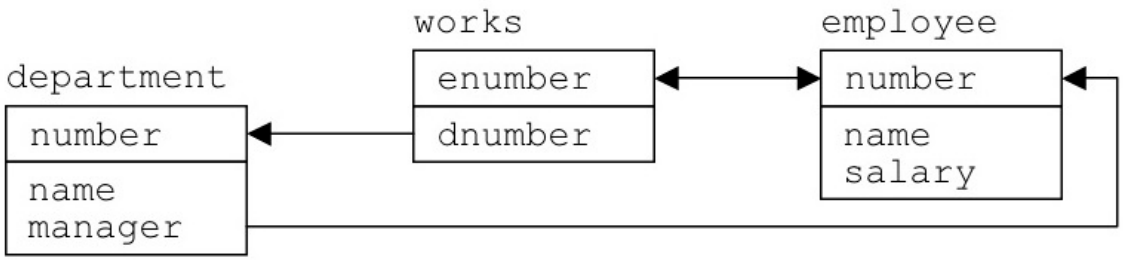
1. *Employees can be identified by their number.*

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

2. *Departments can also be identified by their number.*

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

ACME Case: Extended OPTION 1



Logical Constraints Σ

1. *Employees can be identified by their number.*

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

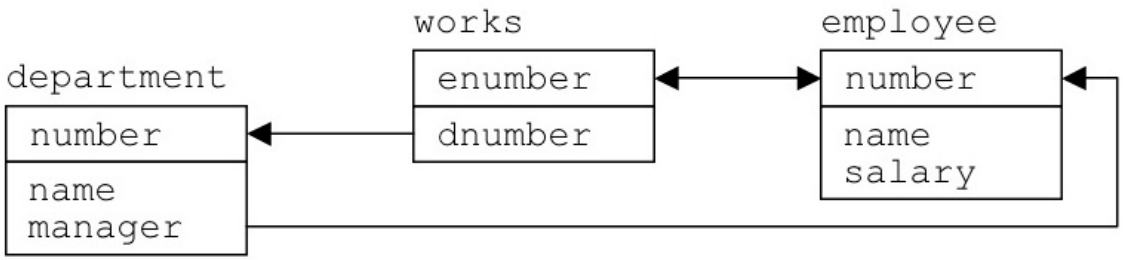
2. *Departments can also be identified by their number.*

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

3. *At most one working relationship to a department can exist for a given employee.*

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

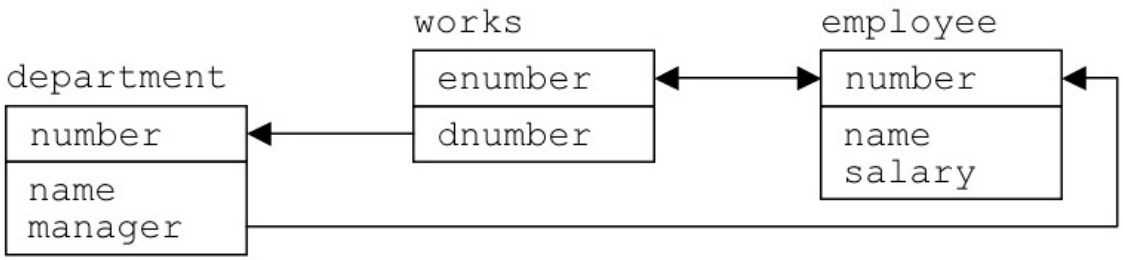
ACME Case: Extended OPTION 1



4. *Only employees can be the manager of a department.*

$$\forall x.(\exists y, z.\texttt{department}(y, z, x) \rightarrow \exists u, v.\texttt{employee}(x, u, v))$$

ACME Case: Extended OPTION 1



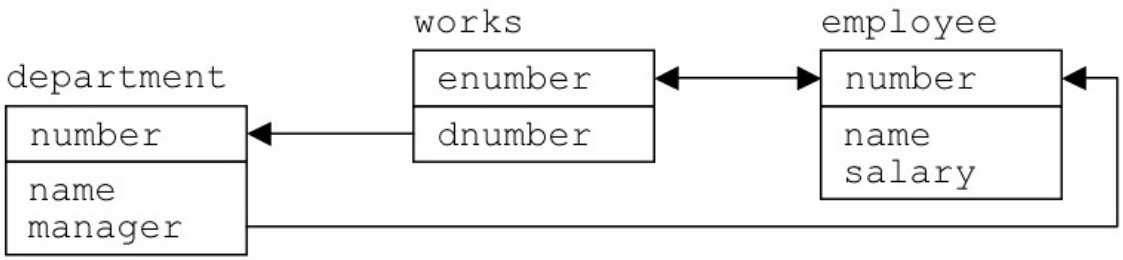
4. *Only employees can be the manager of a department.*

$$\forall x.(\exists y, z.\texttt{department}(y, z, x) \rightarrow \exists u, v.\texttt{employee}(x, u, v))$$

5. *Working relationships can only exists between employees and departments.*

$$\forall x, y.(\texttt{works}(x, y) \rightarrow \exists u_1, u_2, v_1, v_2.(\texttt{employee}(x, u_1, v_1) \wedge \texttt{department}(y, u_2, v_2)))$$

ACME Case: Extended OPTION 1



4. *Only employees can be the manager of a department.*

$$\forall x.(\exists y, z.\texttt{department}(y, z, x) \rightarrow \exists u, v.\texttt{employee}(x, u, v))$$

5. *Working relationships can only exists between employees and departments.*

$$\forall x, y.(\texttt{works}(x, y) \rightarrow \exists u_1, u_2, v_1, v_2.(\texttt{employee}(x, u_1, v_1) \wedge \texttt{department}(y, u_2, v_2)))$$

6. *Every employee must work.*

$$\forall x.(\exists y, z.\texttt{employee}(x, y, z) \rightarrow \exists u.\texttt{works}(x, u))$$

References, Pointers and Linked Structures

One of the most powerful idioms of physical design derives from underlying data store based on the RAM model.

References, Pointers and Linked Structures

One of the most powerful idioms of physical design derives from underlying data store based on the RAM model.

Access to records located in RAM requires *constant time* when search keys correspond to pointer values that *address* or *reference* the store.

References, Pointers and Linked Structures

One of the most powerful idioms of physical design derives from underlying data store based on the RAM model.

Access to records located in RAM requires *constant time* when search keys correspond to pointer values that *address* or *reference* the store.[1]

[1] Dereferencing a pointer value is the ultimate means of data access in the RAM model.

References, Pointers and Linked Structures

One of the most powerful idioms of physical design derives from underlying data store based on the RAM model.

Access to records located in RAM requires *constant time* when search keys correspond to pointer values that *address* or *reference* the store.[1]

Idiom applies in several circumstances.

- Main memory data.

---

[1] Dereferencing a pointer value is the ultimate means of data access in the RAM model.

References, Pointers and Linked Structures

One of the most powerful idioms of physical design derives from underlying data store based on the RAM model.

Access to records located in RAM requires *constant time* when search keys correspond to pointer values that *address* or *reference* the store.[1]

Idiom applies in several circumstances.

- Main memory data.
- Data stored on external devices that provide close to random access: magnetic discs, flash memory, etc.

---

[1] Dereferencing a pointer value is the ultimate means of data access in the RAM model.

References, Pointers and Linked Structures

Model this idiom as follows: assume data is organized in *records* of the following form that are located at a particular *address* in the RAM storage.

```
record r of
  field-type_1 f_1
    ⋮
  field-type_k f_k
```

References, Pointers and Linked Structures

Model this idiom as follows: assume data is organized in *records* of the following form that are located at a particular *address* in the RAM storage.

```
record r of
  field-type_1 f_1
    ⋮
  field-type_k f_k
```

Field types can the following.

1. Atomic values, e.g., integers and strings.

References, Pointers and Linked Structures

Model this idiom as follows: assume data is organized in *records* of the following form that are located at a particular *address* in the RAM storage.

```
record r of
  field-type_1 f_1
    ⋮
  field-type_k f_k
```

Field types can the following.

1. Atomic values, e.g., integers and strings.
2. *Reference* or *pointer* values, called *addresses*, that encode locations of records in RAM store.

References, Pointers and Linked Structures

Model this idiom as follows: assume data is organized in *records* of the following form that are located at a particular *address* in the RAM storage.

$$\begin{array}{l}\texttt{record r of}\\ \quad \texttt{field-type}_1\ \texttt{f}_1\\ \quad\ \vdots\\ \quad \texttt{field-type}_k\ \texttt{f}_k\end{array}$$

Field types can the following.

1. Atomic values, e.g., integers and strings.
2. *Reference* or *pointer* values, called *addresses*, that encode locations of records in RAM store.

Records are captured by a collection of access paths.

$$\{\texttt{r-f}_1/2/1, \ldots, \texttt{r-f}_k/2/1\}$$

References, Pointers and Linked Structures

Intended meaning of the following atomic query plan is to interpret the binding for variable x_1 as an address of an "r" record.

$$\mathtt{r}\text{-}\mathtt{f}_i(x_1, x_2)$$

References, Pointers and Linked Structures

Intended meaning of the following atomic query plan is to interpret the binding for variable x_1 as an address of an "r" record.

$$\mathtt{r}\text{-}\mathtt{f}_i(x_1, x_2)$$

When executed, RAM store is accessed using this address to retrieve the value of field $\mathtt{f}_i$ into variable x_2.

References, Pointers and Linked Structures

Intended meaning of the following atomic query plan is to interpret the binding for variable x_1 as an address of an "r" record.

$$\texttt{r-f}_i(x_1, x_2)$$

When executed, RAM store is accessed using this address to retrieve the value of field $\texttt{f}_i$ into variable x_2.

With respect to our `first/next` protocol, pseudocode that reflects this intended meaning is given as follows.

```
function r-fi-first
   x2 := x1 -> fi
   return true

function r-fi-next
   return false
```

References, Pointers and Linked Structures

Intended meaning of the following atomic query plan is to interpret the binding for variable x_1 as an address of an "r" record.

$$\texttt{r-f}_i(x_1, x_2)$$

When executed, RAM store is accessed using this address to retrieve the value of field $\texttt{f}_i$ into variable x_2.

With respect to our `first/next` protocol, pseudocode that reflects this intended meaning is given as follows.

```
function r-fi-first
  x2 := x1 -> fi
  return true

function r-fi-next
  return false
```

Also assume each access path is associated with a constraint ensuring the operation of extracting a field from a record is a function.

$$\forall x, y_1, y_2.((\texttt{r-f}_i(x, y_1) \wedge \texttt{r-f}_i(x, y_2)) \rightarrow (y_1 \approx y_2))$$

ACME Case: Extended Option 1 Physical Design

Represent individual `employee` and `department` entities as records conforming to the following `emp` and `dept` declarations.

```
record emp of
     integer    num
     string     name
     integer    salary
     reference  dept
```

```
record dept of
     integer    num
     string     name
     reference  manager
```

ACME Case: Extended Option 1 Physical Design

Represent individual `employee` and `department` entities as records conforming to the following `emp` and `dept` declarations.

```
record emp of
     integer    num
     string     name
     integer    salary
     reference  dept
```

```
record dept of
     integer    num
     string     name
     reference  manager
```

Then organize the `emp` records in a collection data structure (such as array or linked list) called `empfile`.

ACME Case: Extended Option 1 Physical Design

Represent individual `employee` and `department` entities as records conforming to the following `emp` and `dept` declarations.

```
record emp of
     integer    num
     string     name
     integer    salary
     reference  dept
```

```
record dept of
     integer    num
     string     name
     reference  manager
```

Then organize the `emp` records in a collection data structure (such as array or linked list) called `empfile`.[1]

---

[1] These data structures allow us to scan the addresses of all `emp` records that represent `employee` entities.

ACME Case: Extended Option 1 Physical Design

A physical design for this data encoding is given as follows.

Physical Signature

$$S_A = \{ \texttt{empfile}/1/0, \texttt{emp-num}/2/1, \texttt{emp-name}/2/1, \texttt{emp-salary}/2/1, \texttt{emp-dept}/2/1, \texttt{dept-num}/2/1, \texttt{dept-name}/2/1, \texttt{dept-manager}/2/1 \}$$

ACME Case: Extended Option 1 Physical Design

A physical design for this data encoding is given as follows.

Physical Signature

$$S_A = \{ \begin{array}{l} \texttt{empfile}/1/0, \\ \texttt{emp-num}/2/1, \texttt{emp-name}/2/1, \texttt{emp-salary}/2/1, \texttt{emp-dept}/2/1, \\ \texttt{dept-num}/2/1, \texttt{dept-name}/2/1, \texttt{dept-manager}/2/1 \end{array} \}$$

Physical Constraints Σ' (a selection)

1. *Every* `emp` *record has all its fields*.

$$\begin{array}{ll} (a) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-num}(x, y)) \\ (b) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-name}(x, y)) \\ (c) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-salary}(x, y)) \\ (d) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-dept}(x, y)) \end{array}$$

ACME Case: Extended Option 1 Physical Design

A physical design for this data encoding is given as follows.

Physical Signature

$$S_A = \{ \begin{array}{l} \texttt{empfile}/1/0, \\ \texttt{emp-num}/2/1, \texttt{emp-name}/2/1, \texttt{emp-salary}/2/1, \texttt{emp-dept}/2/1, \\ \texttt{dept-num}/2/1, \texttt{dept-name}/2/1, \texttt{dept-manager}/2/1 \end{array} \}$$

Physical Constraints Σ' (a selection)

1. *Every* `emp` *record has all its fields*.

$$\begin{array}{ll} (a) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-num}(x,y)) \\ (b) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-name}(x,y)) \\ (c) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-salary}(x,y)) \\ (d) & \forall x.(\texttt{empfile}(x) \rightarrow \exists y.\texttt{emp-dept}(x,y)) \end{array}$$

2. *The range of* `emp-dept` *is a* `dept` *record in* `deptfile`.

$$\forall x, y.(\texttt{emp-dept}(x,y) \rightarrow \texttt{deptfile}(y))$$

ACME Case: Extended Option 1 Physical Design

3. *Every* `dept` *record has all its fields.*

$$
\begin{array}{ll}
(a) & \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-num}(x, y)) \\
(b) & \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-name}(x, y)) \\
(c) & \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-manager}(x, y))
\end{array}
$$

ACME Case: Extended Option 1 Physical Design

3. *Every* `dept` *record has all its fields.*

$$
\begin{array}{ll}
(a) & \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-num}(x, y)) \\
(b) & \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-name}(x, y)) \\
(c) & \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-manager}(x, y))
\end{array}
$$

4. *The range of* `dept-manager` *is an* `emp` *record in* `empfile`.

$$\forall x, y.(\texttt{dept-manager}(x, y) \rightarrow \texttt{empfile}(y))$$

ACME Case: Extended Option 1 Physical Design

3. *Every* `dept` *record has all its fields.*

$$(a) \quad \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-num}(x, y))$$
$$(b) \quad \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-name}(x, y))$$
$$(c) \quad \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-manager}(x, y))$$

4. *The range of* `dept-manager` *is an* `emp` *record in* `empfile`.

$$\forall x, y.(\texttt{dept-manager}(x, y) \rightarrow \texttt{empfile}(y))$$

Mapping Constraints Σ'' (a selection)

1. *For every* `employee` *there is an* `emp` *record.*

$$\forall x.y, z.(\texttt{employee}(x, y, z) \rightarrow \exists w.(\texttt{empfile}(w) \wedge \texttt{emp-num}(w, x)))$$

ACME Case: Extended Option 1 Physical Design

3. *Every* `dept` *record has all its fields.*

$$(a) \quad \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-num}(x, y))$$
$$(b) \quad \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-name}(x, y))$$
$$(c) \quad \forall x.(\texttt{deptfile}(x) \rightarrow \exists y.\texttt{dept-manager}(x, y))$$

4. *The range of* `dept-manager` *is an* `emp` *record in* `empfile`.

$$\forall x, y.(\texttt{dept-manager}(x, y) \rightarrow \texttt{empfile}(y))$$

Mapping Constraints Σ'' (a selection)

1. *For every* `employee` *there is an* `emp` *record.*

$$\forall x.y, z.(\texttt{employee}(x, y, z) \rightarrow \exists w.(\texttt{empfile}(w) \wedge \texttt{emp-num}(w, x)))$$

2. *The contents of the* `name` *field of an* `emp` *record corresponding to a particular* `employee` *contains this employee's name.*

$$\forall x, y, z, w.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x)) \rightarrow \texttt{emp-name}(w, y))$$

ACME Case: Extended Option 1 Physical Design

3. *Same for the* `salary` *field.*

$$\forall x, y, z, w.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x)) \rightarrow \texttt{emp-salary}(w, z))$$

ACME Case: Extended Option 1 Physical Design

3. *Same for the* `salary` *field.*

$$\forall x, y, z, w.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x)) \rightarrow \texttt{emp-salary}(w, z))$$

4. *The contents of the* `dept` *field of an* `emp` *record is a reference to the* `dept` *record that represents the* `department` *for which this employee* `works`.

$$\forall x, y, z, w, u, v.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x) \wedge \texttt{works}(x, u) \wedge \texttt{dept-num}(v, u)) \rightarrow \texttt{emp-dept}(w, v))$$

ACME Case: Extended Option 1 Physical Design

3. *Same for the* `salary` *field.*

$$\forall x, y, z, w.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x)) \rightarrow \texttt{emp-salary}(w, z))$$

4. *The contents of the* `dept` *field of an* `emp` *record is a reference to the* `dept` *record that represents the* `department` *for which this employee* `works`.

$$\forall x, y, z, w, u, v.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x) \wedge \texttt{works}(x, u) \wedge \texttt{dept-num}(v, u)) \rightarrow \texttt{emp-dept}(w, v))$$

5. *Every* `emp` *record in* `empfile` *represents an* `employee`.

$$\forall x, y, z, w.((\texttt{empfile}(w) \wedge \texttt{emp-num}(w, x) \wedge \texttt{emp-name}(w, y) \wedge \texttt{emp-salary}(w, z)) \rightarrow \texttt{employee}(x, y, z))$$

ACME Case: Extended Option 1 Physical Design

3. *Same for the* `salary` *field.*

$$\forall x, y, z, w.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x)) \rightarrow \texttt{emp-salary}(w, z))$$

4. *The contents of the* `dept` *field of an* `emp` *record is a reference to the* `dept` *record that represents the* `department` *for which this employee* `works`.

$$\forall x, y, z, w, u, v.((\texttt{employee}(x, y, z) \wedge \texttt{emp-num}(w, x) \wedge \texttt{works}(x, u) \wedge \texttt{dept-num}(v, u)) \rightarrow \texttt{emp-dept}(w, v))$$

5. *Every* `emp` *record in* `empfile` *represents an* `employee`.

$$\forall x, y, z, w.((\texttt{empfile}(w) \wedge \texttt{emp-num}(w, x) \wedge \texttt{emp-name}(w, y) \wedge \texttt{emp-salary}(w, z)) \rightarrow \texttt{employee}(x, y, z))$$

6. *Every* `emp` *record and the associated* `dept` *record referenced by the* `dept` *field represents a* `works` *relationship.*

$$\forall x, y, v, w.((\texttt{empfile}(w) \wedge \texttt{emp-num}(w, x) \wedge \texttt{emp-dept}(w, v) \wedge \texttt{dept-num}(v, y)) \rightarrow \texttt{works}(x, y))$$

ACME Case: Extended Option 1 Physical Design

7. *For every* `department` *there is a* `dept` *record.*

$$\forall x, y, z.(\texttt{department}(x, y, z) \rightarrow \exists w.(\texttt{deptfile}(w) \wedge \texttt{dept-num}(w, x)))$$

ACME Case: Extended Option 1 Physical Design

7. *For every* `department` *there is a* `dept` *record.*

$$\forall x, y, z.(\texttt{department}(x, y, z) \rightarrow \exists w.(\texttt{deptfile}(w) \wedge \texttt{dept-num}(w, x)))$$

8. *Every* `dept` *record has the proper contents for its* `name` *field.*

$$\forall x, y, z, w.((\texttt{department}(x, y, z) \wedge \texttt{dept-num}(w, x)) \rightarrow \texttt{dept-name}(w, y))$$

ACME Case: Extended Option 1 Physical Design

7. *For every* `department` *there is a* `dept` *record.*

$$\forall x, y, z.(\texttt{department}(x, y, z) \rightarrow \exists w.(\texttt{deptfile}(w) \wedge \texttt{dept-num}(w, x)))$$

8. *Every* `dept` *record has the proper contents for its* `name` *field.*

$$\forall x, y, z, w.((\texttt{department}(x, y, z) \wedge \texttt{dept-num}(w, x)) \rightarrow \texttt{dept-name}(w, y))$$

9. *Same for its* `manager` *field.*

$$\begin{aligned}\forall x, y, z, w.((&\texttt{department}(x, y, z) \wedge \texttt{dept-num}(w, x)) \\ &\rightarrow \exists v.(\texttt{dept-manager}(w, v) \wedge \texttt{emp-num}(v, z)))\end{aligned}$$

ACME Case: Extended Option 1 Physical Design

7. *For every* `department` *there is a* `dept` *record.*

$$\forall x, y, z.(\texttt{department}(x, y, z) \rightarrow \exists w.(\texttt{deptfile}(w) \wedge \texttt{dept-num}(w, x)))$$

8. *Every* `dept` *record has the proper contents for its* `name` *field.*

$$\forall x, y, z, w.((\texttt{department}(x, y, z) \wedge \texttt{dept-num}(w, x)) \rightarrow \texttt{dept-name}(w, y))$$

9. *Same for its* `manager` *field.*

$$\begin{aligned}\forall x, y, z, w.((&\texttt{department}(x, y, z) \wedge \texttt{dept-num}(w, x)) \\ &\rightarrow \exists v.(\texttt{dept-manager}(w, v) \wedge \texttt{emp-num}(v, z)))\end{aligned}$$

10. *Every* `dept` *record in the* `deptfile` *represents a* `department`.

$$\begin{aligned}\forall x, y, z, w, v.((&\texttt{dept-num}(w, x) \wedge \texttt{dept-name}(w, y) \wedge \texttt{dept-manager}(w, v) \wedge \texttt{emp-num}(v, z)) \\ &\rightarrow \texttt{department}(x, y, z))\end{aligned}$$

ACME Case: Extended Option 1 Queries and Plans

ACME's APS department specifies a user query Q to *list employee numbers, names and department names for all employees*.

$$\exists y, v, w.(\texttt{employee}(x_1, x_2, y) \wedge \texttt{works}(x_1, v) \wedge \texttt{department}(v, x_3, w))$$

ACME Case: Extended Option 1 Queries and Plans

ACME's APS department specifies a user query Q to *list employee numbers, names and department names for all employees*.

$$\exists y, v, w.(\texttt{employee}(x_1, x_2, y) \wedge \texttt{works}(x_1, v) \wedge \texttt{department}(v, x_3, w))$$

A query plan Q' that implements this user query with respect to the above physical design is defined as follows.

$$\{\exists e, d.(\texttt{empfile}(e) \wedge \texttt{emp-num}(e, x_1) \wedge \texttt{emp-name}(e, x_2) \\ \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, x_3))\}$$

ACME Case: Extended Option 1 Queries and Plans

ACME's APS department specifies a user query Q to *list employee numbers, names and department names for all employees*.

$$\exists y, v, w.(\texttt{employee}(x_1, x_2, y) \wedge \texttt{works}(x_1, v) \wedge \texttt{department}(v, x_3, w))$$

A query plan Q' that implements this user query with respect to the above physical design is defined as follows.

$$\{\exists e, d.(\texttt{empfile}(e) \wedge \texttt{emp-num}(e, x_1) \wedge \texttt{emp-name}(e, x_2) \\ \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, x_3))\}$$

Observations

1. $\Sigma \models Q \lhd Q'$, where Σ consists of the above physical design.

ACME Case: Extended Option 1 Queries and Plans

ACME's APS department specifies a user query Q to *list employee numbers, names and department names for all employees*.

$$\exists y, v, w.(\texttt{employee}(x_1, x_2, y) \wedge \texttt{works}(x_1, v) \wedge \texttt{department}(v, x_3, w))$$

A query plan Q' that implements this user query with respect to the above physical design is defined as follows.

$$\{\exists e, d.(\texttt{empfile}(e) \wedge \texttt{emp-num}(e, x_1) \wedge \texttt{emp-name}(e, x_2) \\ \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, x_3))\}$$

Observations

1. $\Sigma \models Q \lhd Q'$, where Σ consists of the above physical design.
2. Nested loops joins combined with the pseudocode for the field extraction access paths yield the desired plan.

ACME Case: Extended Option 1 Queries and Plans

ACME's APS department specifies a user query Q to *list employee numbers, names and department names for all employees*.

$$\exists y, v, w.(\texttt{employee}(x_1, x_2, y) \wedge \texttt{works}(x_1, v) \wedge \texttt{department}(v, x_3, w))$$

A query plan Q' that implements this user query with respect to the above physical design is defined as follows.

$$\{\exists e, d.(\texttt{empfile}(e) \wedge \texttt{emp-num}(e, x_1) \wedge \texttt{emp-name}(e, x_2) \\ \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, x_3))\}$$

Observations

1. $\Sigma \models Q \lhd Q'$, where Σ consists of the above physical design.
2. Nested loops joins combined with the pseudocode for the field extraction access paths yield the desired plan.
3. Once a reference to an employee record is obtained, all emaining operations are dereferencing operations that run in $O(1)$ in the RAM model.

ACME Case: Extended Option 1 Queries and Plans

Consider the following.

- A functional constraint in the physical design stating that the `emp` records correspond *one-to-one* to `employee` entities.

$$\forall x, y_1, y_2.(\texttt{emp-num}(y_1, x) \wedge \texttt{emp-num}(y_2, x)) \rightarrow (y_1 \approx y_2)$$

- Constraints expressing the functionality of field access.

ACME Case: Extended Option 1 Queries and Plans

Consider the following.

- A functional constraint in the physical design stating that the `emp` records correspond *one-to-one* to `employee` entities.

$$\forall x, y_1, y_2.(\texttt{emp-num}(y_1, x) \wedge \texttt{emp-num}(y_2, x)) \rightarrow (y_1 \approx y_2)$$

- Constraints expressing the functionality of field access.

Altogether implies that the following query plan also implements the query.

$$\exists e, d.(\texttt{empfile}(e) \wedge \texttt{emp-num}(e, x_1) \wedge \texttt{emp-name}(e, x_2) \\ \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, x_3))$$

ACME Case: Extended Option 1 Queries and Plans

Consider the following.

- A functional constraint in the physical design stating that the `emp` records correspond *one-to-one* to `employee` entities.

$$\forall x, y_1, y_2.(\texttt{emp-num}(y_1, x) \wedge \texttt{emp-num}(y_2, x)) \rightarrow (y_1 \approx y_2)$$

- Constraints expressing the functionality of field access.

Altogether implies that the following query plan also implements the query.

$$\exists e, d.(\texttt{empfile}(e) \wedge \texttt{emp-num}(e, x_1) \wedge \texttt{emp-name}(e, x_2) \\ \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, x_3))$$

Observations

1. The final duplicate elimination operation has been omitted.

ACME Case: Extended Option 1 Queries and Plans

Consider the following.

- A functional constraint in the physical design stating that the `emp` records correspond *one-to-one* to `employee` entities.

$$\forall x, y_1, y_2.(\texttt{emp-num}(y_1, x) \wedge \texttt{emp-num}(y_2, x)) \rightarrow (y_1 \approx y_2)$$

- Constraints expressing the functionality of field access.

Altogether implies that the following query plan also implements the query.

$$\exists e, d.(\texttt{empfile}(e) \wedge \texttt{emp-num}(e, x_1) \wedge \texttt{emp-name}(e, x_2) \\ \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, x_3))$$

Observations

1. The final duplicate elimination operation has been omitted.
2. The plan runs in $O(n)$ time, where n is the number of employee entities (recall that the underlying data structure with `emp` records is an array or linked list.)

Exercise: Listing Departments

- another user query: `department`(x, y, z)

  $\Rightarrow$ remember that we *do not* have an access path that *scans* departments

Exercise: Listing Departments

- another user query: `department`(x, y, z)

  $\Rightarrow$ remember that we *do not* have an access path that *scans* departments

  $\Rightarrow$ can we find a plan?

Exercise: Listing Departments

- another user query: $\texttt{department}(x, y, z)$

  $\Rightarrow$ remember that we *do not* have an access path that *scans* departments

  $\Rightarrow$ can we find a plan? NO

- add a constraint "*every boss works for his/her department*":

$$\forall x, y, z.\texttt{department}(x, y, z) \rightarrow \texttt{works}(z, x)$$

Exercise: Listing Departments

- another user query: $\texttt{department}(x, y, z)$

  $\Rightarrow$ remember that we *do not* have an access path that *scans* departments

  $\Rightarrow$ can we find a plan? NO

- add a constraint "*every boss works for his/her department*":

$$\forall x, y, z.\texttt{department}(x, y, z) \rightarrow \texttt{works}(z, x)$$

- Attempt 1:

$$\{\exists e, d, m.\ \texttt{empfile}(e) \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-num}(d, x) \wedge \texttt{dept-name}(d, y) \wedge \texttt{dept-manager}(d, m) \wedge \texttt{emp-num}(m, z) \}$$

Exercise: Listing Departments

- another user query: $\texttt{department}(x, y, z)$

  $\Rightarrow$ remember that we *do not* have an access path that *scans* departments

  $\Rightarrow$ can we find a plan? NO

- add a constraint "*every boss works for his/her department*":

$$\forall x, y, z.\texttt{department}(x, y, z) \rightarrow \texttt{works}(z, x)$$

- Attempt 1:

$$\{\exists e, d, m.\ \texttt{empfile}(e) \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-num}(d, x) \wedge \texttt{dept-name}(d, y) \wedge \texttt{dept-manager}(d, m) \wedge \texttt{emp-num}(m, z) \}$$

- Attempt 2:

$$\exists e, d, m.\ \texttt{empfile}(e) \wedge \texttt{emp-num}(e, z) \wedge \texttt{emp-dept}(e, d) \wedge \texttt{dept-name}(d, y) \wedge \texttt{dept-num}(d, x) \wedge \texttt{dept-manager}(d, m) \wedge \texttt{compare}(e, m)$$

ACME Case: Nested Loops and Secondary Indices

Assume ACME's APP department specifies a user query that *list pairs of employee numbers of employees with the same name*.

$$\exists y, z, w.(\texttt{employee}(x_1, y, z) \wedge \texttt{employee}(x_2, y, w)).$$

ACME Case: Nested Loops and Secondary Indices

Assume ACME's APP department specifies a user query that *list pairs of employee numbers of employees with the same name*.

$$\exists y, z, w.(\texttt{employee}(x_1, y, z) \wedge \texttt{employee}(x_2, y, w)).$$

The following is a query plan implementing the query.

$$\begin{aligned}\exists y, z, w, v.(&\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y)\\ &\wedge \texttt{empfile}(w) \wedge \texttt{emp-num}(w, x_2) \wedge \texttt{emp-name}(w, z)\\ &\wedge \texttt{compare}(y, z))\end{aligned}$$

ACME Case: Nested Loops and Secondary Indices

Assume ACME's APP department specifies a user query that *list pairs of employee numbers of employees with the same name*.

$$\exists y, z, w.(\texttt{employee}(x_1, y, z) \wedge \texttt{employee}(x_2, y, w)).$$

The following is a query plan implementing the query.

$$\begin{aligned}\exists y, z, w, v.(&\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y)\\ &\wedge \texttt{empfile}(w) \wedge \texttt{emp-num}(w, x_2) \wedge \texttt{emp-name}(w, z)\\ &\wedge \texttt{compare}(y, z))\end{aligned}$$

The plan is not desirable since it considers *all pairs of employees* and then selects those with the same name.

ACME Case: Nested Loops and Secondary Indices

Assume ACME's APP department specifies a user query that *list pairs of employee numbers of employees with the same name*.

$$\exists y, z, w.(\texttt{employee}(x_1, y, z) \wedge \texttt{employee}(x_2, y, w)).$$

The following is a query plan implementing the query.

$$\begin{aligned}\exists y, z, w, v.(&\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y)\\ &\wedge \texttt{empfile}(w) \wedge \texttt{emp-num}(w, x_2) \wedge \texttt{emp-name}(w, z)\\ &\wedge \texttt{compare}(y, z))\end{aligned}$$

The plan is not desirable since it considers *all pairs of employees* and then selects those with the same name.[1]

---

[1] An additional *access path* that allows us to efficiently find references to employee records given an employee *name* would enable plans for the query that avoid this.

ACME Case: Nested Loops and Secondary Indices

Consider adding the following access path and constraint.

$$\texttt{empidx-name}/2/1$$

$$\forall x, y.(\texttt{eidx-name}(x, y) \equiv (\texttt{empfile}(y) \wedge \texttt{emp-name}(y, x)))$$

ACME Case: Nested Loops and Secondary Indices

Consider adding the following access path and constraint.

$$\texttt{empidx-name}/2/1$$

$$\forall x, y.(\texttt{eidx-name}(x, y) \equiv (\texttt{empfile}(y) \wedge \texttt{emp-name}(y, x)))$$

An alternative plan implementing the query that avoids the need to compare all pairs of employees is now possible.

$$\exists y, w, v.(\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y) \\ \wedge \texttt{empidx-name}(y, w) \wedge \texttt{emp-num}(w, x_2))$$

ACME Case: Nested Loops and Secondary Indices

Consider adding the following access path and constraint.

$$\texttt{empidx-name}/2/1$$

$$\forall x, y.(\texttt{eidx-name}(x, y) \equiv (\texttt{empfile}(y) \wedge \texttt{emp-name}(y, x)))$$

An alternative plan implementing the query that avoids the need to compare all pairs of employees is now possible.

$$\exists y, w, v.(\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y) \\ \wedge \texttt{empidx-name}(y, w) \wedge \texttt{emp-num}(w, x_2))$$

Observations

1. Have *stored* references to records in the search structure.

ACME Case: Nested Loops and Secondary Indices

Consider adding the following access path and constraint.

$$\texttt{empidx-name}/2/1$$

$$\forall x, y.(\texttt{eidx-name}(x, y) \equiv (\texttt{empfile}(y) \wedge \texttt{emp-name}(y, x)))$$

An alternative plan implementing the query that avoids the need to compare all pairs of employees is now possible.

$$\exists y, w, v.(\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y) \\ \wedge \texttt{empidx-name}(y, w) \wedge \texttt{emp-num}(w, x_2))$$

Observations

1. Have *stored* references to records in the search structure.
2. Could also have embedded `emp` records in the search structure.

ACME Case: Nested Loops and Secondary Indices

Consider adding the following access path and constraint.

$$\texttt{empidx-name}/2/1$$

$$\forall x, y.(\texttt{eidx-name}(x, y) \equiv (\texttt{empfile}(y) \wedge \texttt{emp-name}(y, x)))$$

An alternative plan implementing the query that avoids the need to compare all pairs of employees is now possible.

$$\exists y, w, v.(\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y) \\ \wedge \texttt{empidx-name}(y, w) \wedge \texttt{emp-num}(w, x_2))$$

Observations

1. Have *stored* references to records in the search structure.
2. Could also have embedded `emp` records in the search structure.

This difference is the basis for classifying indexes as *primary* or *secondary*.

ACME Case: Nested Loops and Secondary Indices

Consider adding the following access path and constraint.

$$\texttt{empidx-name}/2/1$$

$$\forall x, y.(\texttt{eidx-name}(x, y) \equiv (\texttt{empfile}(y) \wedge \texttt{emp-name}(y, x)))$$

An alternative plan implementing the query that avoids the need to compare all pairs of employees is now possible.

$$\exists y, w, v.(\texttt{empfile}(v) \wedge \texttt{emp-num}(v, x_1) \wedge \texttt{emp-name}(v, y) \\ \wedge \texttt{empidx-name}(y, w) \wedge \texttt{emp-num}(w, x_2))$$

Observations

1. Have *stored* references to records in the search structure.
2. Could also have embedded `emp` records in the search structure.

This difference is the basis for classifying indexes as *primary* or *secondary*. Allowing references to be first-class citizens in the physical design removes any need for the latter notion.

ACME Case: Optional Department Assignments

Consider removing the following constraint from the extended OPTION 1 design of PAYROLL.

$$\forall x, y, z.(\texttt{employee}(x, y, z) \rightarrow \exists u.\texttt{works}(x, u))$$

The change means that employees may not be associated with any department, a relaxed design in which each employee now works for *at most one* department.

ACME Case: Optional Department Assignments

Consider removing the following constraint from the extended OPTION 1 design of PAYROLL.

$$\forall x, y, z.(\texttt{employee}(x, y, z) \rightarrow \exists u.\texttt{works}(x, u))$$

The change means that employees may not be associated with any department, a relaxed design in which each employee now works for *at most one* department.

Goal: To accommodate this change in requirements *without* the need for a major modification of the physical design.

ACME Case: Optional Department Assignments

Consider removing the following constraint from the extended OPTION 1 design of PAYROLL.

$$\forall x, y, z.(\texttt{employee}(x, y, z) \rightarrow \exists u.\texttt{works}(x, u))$$

The change means that employees may not be associated with any department, a relaxed design in which each employee now works for *at most one* department.

Goal: To accommodate this change in requirements *without* the need for a major modification of the physical design.

1. Employees and departments continue to be represented by the `emp` and `dept` records.

ACME Case: Optional Department Assignments

Consider removing the following constraint from the extended OPTION 1 design of PAYROLL.

$$\forall x, y, z.(\texttt{employee}(x, y, z) \rightarrow \exists u.\texttt{works}(x, u))$$

The change means that employees may not be associated with any department, a relaxed design in which each employee now works for *at most one* department.

Goal: To accommodate this change in requirements *without* the need for a major modification of the physical design.

1. Employees and departments continue to be represented by the `emp` and `dept` records.
2. *Signal* that an employee is *not* associated with a department by filling the `dept` field of the associated `emp` record with some `null` value.

ACME Case: Optional Department Assignments

To model this situation, we alter the physical design with constraints relating to this `null` value.

Modified Physical Constraints

1. *The* `dept` *field of an* `emp` *record either contains a* `null` *value or a non-*`null` *reference to a* `dept` *record representing a department.*

$$\forall x, y.(\texttt{emp-dept}(x, y) \rightarrow (\texttt{null}(y) \vee \exists z.\texttt{notnull}(y, z)))$$

$$\forall x, y, z.((\texttt{emp-dept}(x, y) \wedge \texttt{notnull}(y, z)) \rightarrow \texttt{deptfile}(z))$$

Nulls Indicating Value Inapplicable

It is a common idiom in a *physical design* to model missing *optional information* using a default `null` value.

The following access path implements a generic test for a `null` reference.

`null`/1/1

Nulls Indicating Value Inapplicable

It is a common idiom in a *physical design* to model missing *optional information* using a default `null` value.

The following access path implements a generic test for a `null` reference.

`null`/1/1

Assumes following pseudocode.

```
function null-first
  if (x1 == NULL) return true
  return false
```

```
function null-next
  return false
```

Nulls Indicating Value Inapplicable

It is a common idiom in a *physical design* to model missing *optional information* using a default `null` value.

The following access path implements a generic test for a `null` reference.

$$\texttt{null}/1/1$$

Assumes following pseudocode.

```
function null-first
  if (x1 == NULL) return true
  return false
```

```
function null-next
  return false
```

Also assume any physical design includes the following constraint ensuring there is at most one `NULL` value.

$$\forall x, y.((\texttt{null}(x) \wedge \texttt{null}(y)) \rightarrow (x \approx y))$$

ACME Case: Checking for Nulls

Assume ACME's APP department specifies a user query that *lists numbers and names of all employees that do not work for any department*.

$$\exists y.(\texttt{employee}(x_1, x_2, y) \wedge \neg \exists z.\texttt{works}(x_1, z))$$

ACME Case: Checking for Nulls

Assume ACME's APP department specifies a user query that *lists numbers and names of all employees that do not work for any department*.

$$\exists y.(\texttt{employee}(x_1, x_2, y) \wedge \neg \exists z.\texttt{works}(x_1, z))$$

The following is a query plan implementing the query with respect to the modified physical design.

$$\exists y, z.(\texttt{empfile}(y) \wedge \texttt{emp-num}(y, x_1) \wedge \texttt{emp-name}(y, x_2) \\ \wedge \texttt{emp-dept}(y, z) \wedge \texttt{null}(z))$$

Non-Null Type Casting

The following access path implements a generic type casting for non-null values.

`notnull`/2/1

Non-Null Type Casting

The following access path implements a generic type casting for non-null values.

`notnull/2/1`

Assumes following pseudocode.

```
function notnull-first
  if ($x_1$ <> NULL)
    $x_2$ := $x_1$
    return true
  return false
```

```
function notnull-next
  return false
```

Non-Null Type Casting

The following access path implements a generic type casting for non-null values.

$$\texttt{notnull}/2/1$$

Assumes following pseudocode.

```
function notnull-first
  if ($x_1$ <> NULL)
    $x_2$ := $x_1$
    return true
  return false
```

```
function notnull-next
  return false
```

Also assume any physical design includes the following constraints.

$$\forall x, y, z.((\texttt{notnull}(x, y) \wedge \texttt{notnull}(x, z)) \rightarrow (y \approx z))$$

$$\forall x.((\exists y.\texttt{notnull}(x, y)) \equiv \neg\, \texttt{null}(x))$$

ACME Case: Ensuring Non-Nulls

Assume ACME's APP department specifies a user query that *lists employee numbers, names and department names for all employees (who must now work for a department)*.

$$\exists y, z, w.(\texttt{employee}(x_1, x_2, y) \wedge \texttt{works}(x_1, z) \wedge \texttt{department}(z, x_3, w))$$

ACME Case: Ensuring Non-Nulls

Assume ACME's APP department specifies a user query that *lists employee numbers, names and department names for all employees (who must now work for a department)*.

$$\exists y, z, w.(\texttt{employee}(x_1, x_2, y) \wedge \texttt{works}(x_1, z) \wedge \texttt{department}(z, x_3, w))$$

The following is a query plan implementing the query with respect to the modified physical design.

$$\exists y, z, w.(\texttt{empfile}(y) \wedge \texttt{emp-num}(y, x_1) \wedge \texttt{emp-name}(y, x_2) \\ \wedge \texttt{emp-dept}(y, z) \wedge \texttt{notnull}(z, w) \wedge \texttt{dept-name}(w, x_3))$$

ACME Case: Partitioning Employee Records

Consider the following three constraints.

$$\forall x.(\texttt{empfile-low}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y \leq 30k)))$$

$$\forall x.(\texttt{empfile-mid}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (30k < y) \wedge (y \leq 130k)))$$

$$\forall x.(\texttt{empfile-high}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y < 130k)))$$

ACME Case: Partitioning Employee Records

Consider the following three constraints.

$$\forall x.(\texttt{empfile-low}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y \leq 30k)))$$

$$\forall x.(\texttt{empfile-mid}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (30k < y) \wedge (y \leq 130k)))$$

$$\forall x.(\texttt{empfile-high}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y < 130k)))$$

The constraints partition `emp` records into three disjoint subsets based on the value of the `emp-sal` field.

ACME Case: Partitioning Employee Records

Consider the following three constraints.

$$\forall x.(\texttt{empfile-low}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y \leq 30k)))$$

$$\forall x.(\texttt{empfile-mid}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (30k < y) \wedge (y \leq 130k)))$$

$$\forall x.(\texttt{empfile-high}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y < 130k)))$$

The constraints partition `emp` records into three disjoint subsets based on the value of the `emp-sal` field.

Assumes constants and that integer values come with predicates denoting a linear order "$\leq$" and a strict linear order "$<$" interpreted in the standard way.

ACME Case: Partitioning Employee Records

Consider the following three constraints.

$$\forall x.(\texttt{empfile-low}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y \leq 30k)))$$

$$\forall x.(\texttt{empfile-mid}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (30k < y) \wedge (y \leq 130k)))$$

$$\forall x.(\texttt{empfile-high}(x) \equiv \exists y.(\texttt{empfile}(x) \wedge \texttt{emp-sal}(x, y) \wedge (y < 130k)))$$

The constraints partition `emp` records into three disjoint subsets based on the value of the `emp-sal` field.

Assumes constants and that integer values come with predicates denoting a linear order "$\leq$" and a strict linear order "$<$" interpreted in the standard way.[1]

---

[1] This is a violation of our assumption that signatures are free of constant and function symbols that is easily rectified.

ACME Case: Partitioning Employee Records

Assume the APS department wishes to *list employee numbers for employees making at most* $130k$.

$$\exists y, z.(\texttt{employee}(x_1, y, z) \wedge (z \leq 130k))$$

ACME Case: Partitioning Employee Records

Assume the APS department wishes to *list employee numbers for employees making at most* $130k$.

$$\exists y, z.(\texttt{employee}(x_1, y, z) \wedge (z \leq 130k))$$

Also assume the physical signature for PAYROLL is revised as follows.

- $\{\texttt{empfile}/1\} \subseteq (S_P - S_A)$.
- $\{\texttt{empfile-low}/1/0, \texttt{empfile-mid}/1/0, \texttt{empfile-high}/1/0\} \subseteq S_A$.

ACME Case: Partitioning Employee Records

Assume the APS department wishes to *list employee numbers for employees making at most* $130k$.

$$\exists y, z.(\texttt{employee}(x_1, y, z) \wedge (z \leq 130k))$$

Also assume the physical signature for PAYROLL is revised as follows.

- $\{\texttt{empfile}/1\} \subseteq (S_P - S_A)$.
- $\{\texttt{empfile-low}/1/0, \texttt{empfile-mid}/1/0, \texttt{empfile-high}/1/0\} \subseteq S_A$.

The following plan using concatenation then implements this query.

$$\exists y.((\texttt{empfile-low}(y) \vee \texttt{empfile-mid}(y)) \wedge \texttt{emp-num}(y, x_1))$$

ACME Case: Partitioning Employee Records

An alternative plan using simple complement becomes possible if `empfile` is again made an access path.

- $\{\texttt{empfile}/1/0\} \subseteq S_A$.

ACME Case: Partitioning Employee Records

An alternative plan using simple complement becomes possible if `empfile` is again made an access path.

- $\{\texttt{empfile}/1/0\} \subseteq S_A$.

This alternative plan is then given as follows.

$$\exists y.(\texttt{empfile}(y) \wedge \neg\, \texttt{empfile-high}(y) \wedge \texttt{emp-num}(y, x_1))$$

ACME Case: Partitioning Employee Records

An alternative plan using simple complement becomes possible if `empfile` is again made an access path.

- $\{\texttt{empfile}/1/0\} \subseteq S_A$.

This alternative plan is then given as follows.

$$\exists y.(\texttt{empfile}(y) \wedge \neg\, \texttt{empfile-high}(y) \wedge \texttt{emp-num}(y, x_1))$$

This plan assumes that the partition covers all employee records and that constraints such as

$$\forall x.((x \leq 130k) \rightarrow (x \leq 30k))$$

can also be deduced.

Run-time Typing

Run-time typing allows membership in a subclass to be tested at runtime when given a reference to an object in a superclass.

Run-time Typing

Run-time typing allows membership in a subclass to be tested at runtime when given a reference to an object in a superclass.

A common feature of object-oriented databases that support the definition of elaborate taxonomic knowledge.

Run-time Typing

Run-time typing allows membership in a subclass to be tested at runtime when given a reference to an object in a superclass.

A common feature of object-oriented databases that support the definition of elaborate taxonomic knowledge.

We illustrate how this is accommodated by introducing a notion of ACME employees with high salaries

Run-time Typing

Run-time typing allows membership in a subclass to be tested at runtime when given a reference to an object in a superclass.

A common feature of object-oriented databases that support the definition of elaborate taxonomic knowledge.

We illustrate how this is accommodated by introducing a notion of ACME employees with high salaries

Assume the signature of PAYROLL is revised as follows.

- $\{\texttt{high-employee}/1\} \subseteq S_L$.
- $\{\texttt{emp-high}/1/1\} \subseteq S_A$.

Run-time Typing

Run-time typing allows membership in a subclass to be tested at runtime when given a reference to an object in a superclass.

A common feature of object-oriented databases that support the definition of elaborate taxonomic knowledge.

We illustrate how this is accommodated by introducing a notion of ACME employees with high salaries

Assume the signature of PAYROLL is revised as follows.

- $\{\texttt{high-employee}/1\} \subseteq S_L$.
- $\{\texttt{emp-high}/1/1\} \subseteq S_A$.

The revision modifies the logical design of PAYROLL.

1. Adds a "subclass" of `employee` called `high-employee`.
2. Introduces the access path `emp-high` to enable checking at run-time for membership in `high-employee`.

ACME Case: Highly Paid Employees

Must also add the following constraints are added to Σ.

ACME Case: Highly Paid Employees

Must also add the following constraints are added to Σ.

1. *Highly paid employees are a subset of employees*.

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

ACME Case: Highly Paid Employees

Must also add the following constraints are added to Σ.

1. *Highly paid employees are a subset of employees*.

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

2. *A reference to an* `emp` *record for an employee is also a reference to* `emp` *record for a highly paid employee whenever the reference also qualifies as* `emp-high`.

$$\forall x, y.((\texttt{empfile}(y) \wedge \texttt{emp-high}(y) \wedge \texttt{emp-num}(y, x)) \equiv \texttt{high-employee}(x))$$

ACME Case: Highly Paid Employees

Must also add the following constraints are added to Σ.

1. *Highly paid employees are a subset of employees*.

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

2. *A reference to an* `emp` *record for an employee is also a reference to* `emp` *record for a highly paid employee whenever the reference also qualifies as* `emp-high`.

$$\forall x, y.((\texttt{empfile}(y) \wedge \texttt{emp-high}(y) \wedge \texttt{emp-num}(y, x)) \equiv \texttt{high-employee}(x))$$

E.g.: A user query to *list employee numbers for employees that are highly paid*.

$$\texttt{high-employee}(x)$$

ACME Case: Highly Paid Employees

Must also add the following constraints are added to Σ.

1. *Highly paid employees are a subset of employees*.

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

2. *A reference to an* `emp` *record for an employee is also a reference to* `emp` *record for a highly paid employee whenever the reference also qualifies as* `emp-high`.

$$\forall x, y.((\texttt{empfile}(y) \wedge \texttt{emp-high}(y) \wedge \texttt{emp-num}(y, x)) \equiv \texttt{high-employee}(x))$$

E.g.: A user query to *list employee numbers for employees that are highly paid*.

$$\texttt{high-employee}(x)$$

A query plan that implements this query is given as follows.

$$\exists y.(\texttt{empfile}(y) \wedge \texttt{emp-high}(y) \wedge \texttt{emp-num}(y, x))$$

Pages and Records: Two-level Store

Physical Signature: $\{\texttt{emp-pgscan}/1/0, \texttt{emp-recscan}/2/1\} \subseteq S_A$
($\texttt{empfile}/1/0$ is no longer an access path).

Pages and Records: Two-level Store

Physical Signature: $\{\texttt{emp-pgscan}/1/0, \texttt{emp-recscan}/2/1\} \subseteq S_A$
($\texttt{empfile}/1/0$ is no longer an access path).

Constraints:

1. *Every reference to an* `emp` *record can be obtained by scanning some page of the two-level employee store*:

$$\forall x.(\texttt{empfile}(x) \equiv \exists y.\texttt{emp-recscan}(y, x)).$$

2. *Every record belongs to some page of the two level store*:

$$\forall x, y.(\texttt{emp-recscan}(y, x) \rightarrow \texttt{emp-pgscan}(y)).$$

3. *Every record belongs to a single page in two level store*:

$$\forall x, y_1, y_2.((\texttt{emp-recscan}(y_1, x) \wedge \texttt{emp-recscan}(y_2, x)) \rightarrow (y_1 \approx y_2)).$$

Two-level Store:Queriesand Plans

Query:

$$\exists y, z, w.(\texttt{employee}(x_1, y, z) \wedge \texttt{employee}(x_2, y, w)).$$

Two-level Store:Queriesand Plans

Query:

$$\exists y, z, w.(\texttt{employee}(x_1, y, z) \wedge \texttt{employee}(x_2, y, w)).$$

Plan:

$$\begin{aligned}\exists y, z, w, v, p, q.\ (&\texttt{emppgscan}(p) \wedge \texttt{emppgscan}(q)\\ &\wedge \texttt{emprecscan}(p, y) \wedge \texttt{emp-num}(y, x_1) \wedge \texttt{emp-name}(y, w)\\ &\wedge \texttt{emprecscan}(q, z) \wedge \texttt{emp-num}(z, x_2) \wedge \texttt{emp-name}(z, v)\\ &\wedge \texttt{compare}(w, v)).\end{aligned}$$

Further Topics in Physical Design

The following topics are also discussed in the book.

- How to incorporate user defined functions.

Further Topics in Physical Design

The following topics are also discussed in the book.

- How to incorporate user defined functions.
- Hash-based indexing.

Further Topics in Physical Design

The following topics are also discussed in the book.

- How to incorporate user defined functions.
- Hash-based indexing.
- Two-level references.

Further Topics in Physical Design

The following topics are also discussed in the book.

- How to incorporate user defined functions.
- Hash-based indexing.
- Two-level references.
- Disk-based search structures: the ISAM case.

Further Topics in Physical Design

The following topics are also discussed in the book.

- How to incorporate user defined functions.
- Hash-based indexing.
- Two-level references.
- Disk-based search structures: the ISAM case.
- Further examples relating to ACME's extended PAYROLL system.