Databases Essentials

Exercise solutions

Antonio Albano
Department of Computer Science
University of Pisa

February 13, 2015
Revision, December 1, 2020

CONTENTS

Chapter 2

INFORMATION MODELING

Section 2.4 ODM: An Object Data Model

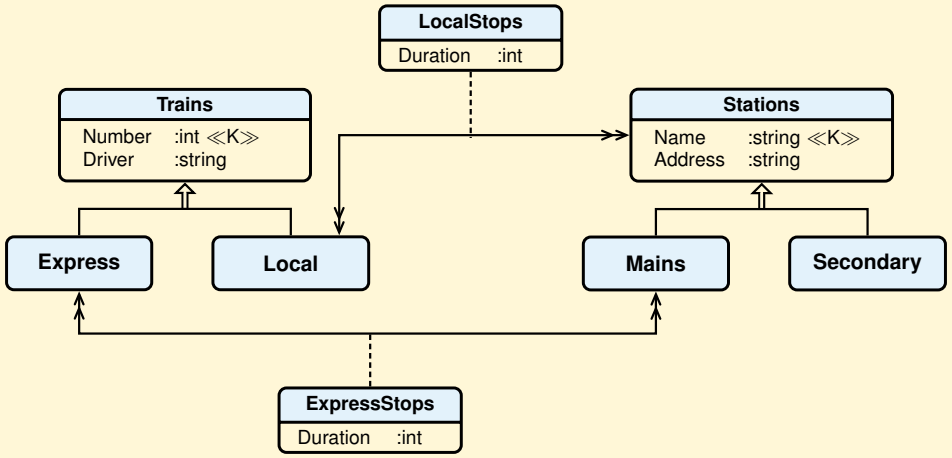
2.4.7 Exercises

1. We would like to design a database for the following facts.

   A train has a unique number and a driver. Trains are either local or express, but never both.

   A station has a unique name and an address. Stations are either main or secondary, but never both. Local trains stop at all stations. Express trains stop only at the main stations. Each stop of a train in a station has a duration. Design a conceptual schema for the database.

Solution

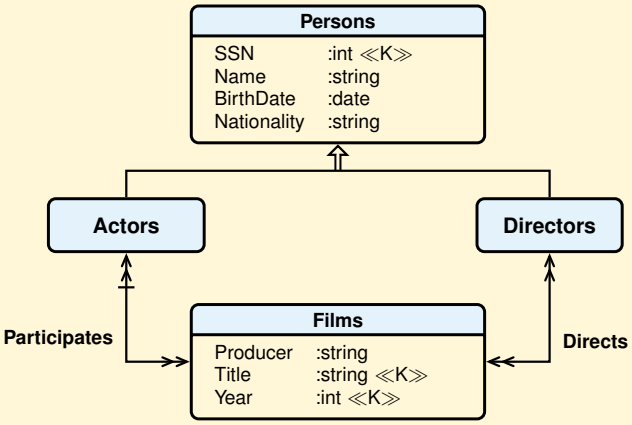


2. Design a conceptual schema for a database to keep track of actors and directors of films.

   Each actor or director has a unique name, a birth year, and a nationality. An actor may be also a director.

   Each film has a title, the production year, the actors, a director, and a producer. Films produced the same year have different titles.

Solution

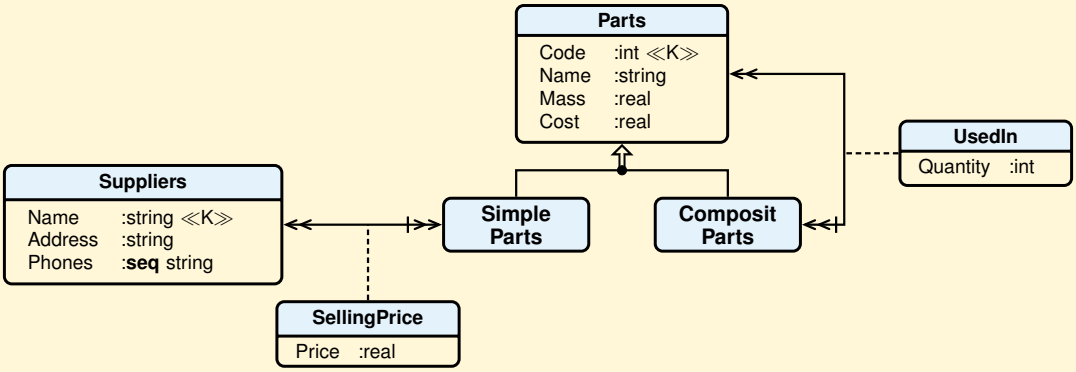


3. Consider the following information about a manufacturing company's parts and suppliers database.

   The database contains information about the way certain parts are manufactured out of other parts: the subparts that are involved in the manufacture of a part, the number of subparts used, the cost of manufacturing a part from its subparts, the mass of the part as result of the subparts assemblage. The manufactured parts may themselves be subparts in a further manufacturing process. In addition, certain information must be held on the parts themselves: their unique code, name and, if they are imported (i.e., manufactured externally), the supplier and the purchase cost.

   Suppliers have an unique name, an address and several phones. Design a conceptual schema for the database.

Solution

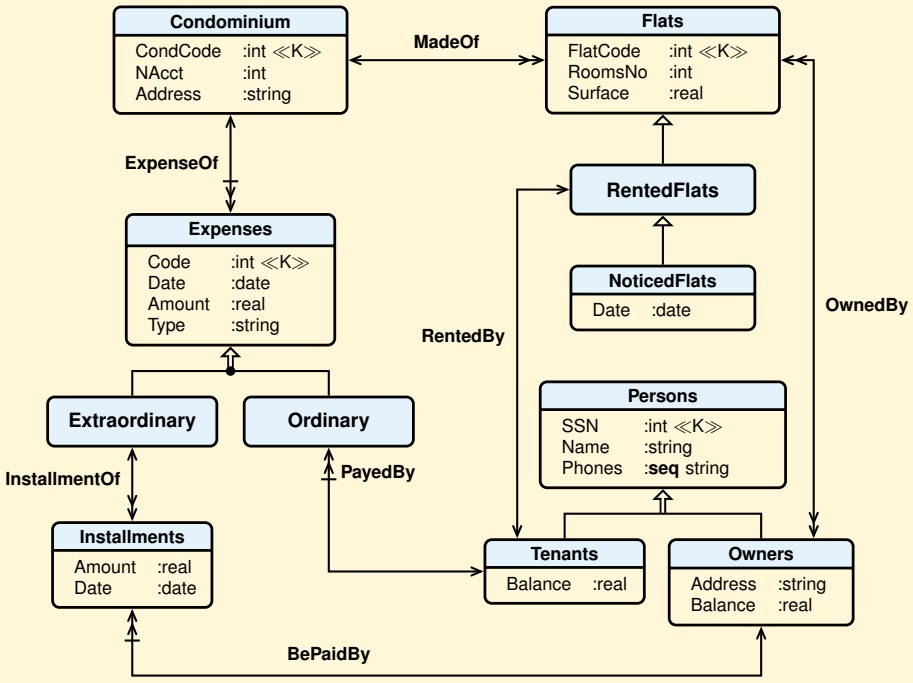


4. We would like to design a database to maintain information for an administrator of condominium (i.e. a block of flats).
   Each condominium has a code (the key), an address and the number of the checking account where should be payed the supported expenses.
   A condominium is made of flats, and we are interested in the flat code (the key), the number rooms, the surface.

   The flats can be rented to a tenant, and we are interested in tenants name, the social-security numbers (the key), the telephones (more than one) and the balance, that is the amount that tenant must pay for running expenses. Some rented flats can have been noticed, and in this case we are interested in the date of the notice.
   A flat can have several owners, and an owner can possess several flats. Of every owner we are interested in the name, the social-security numbers (the key), the address, the telephones (more than one) and the balance, that is the sum that the owner must pay for supported expenses.

   Maintenance expenses for the condominium are described by the code of identification, the type (light, cleaning, elevator, etc), the date and the amount. The expenses are classified as extraordinary, to be paid by owners, or as ordinary to be paid by tenants. Ordinary expenses must be paid in one installment, while extraordinary expenses can be paid in more installments, and for each of them it is necessary to remember the date and the amount.

Solution



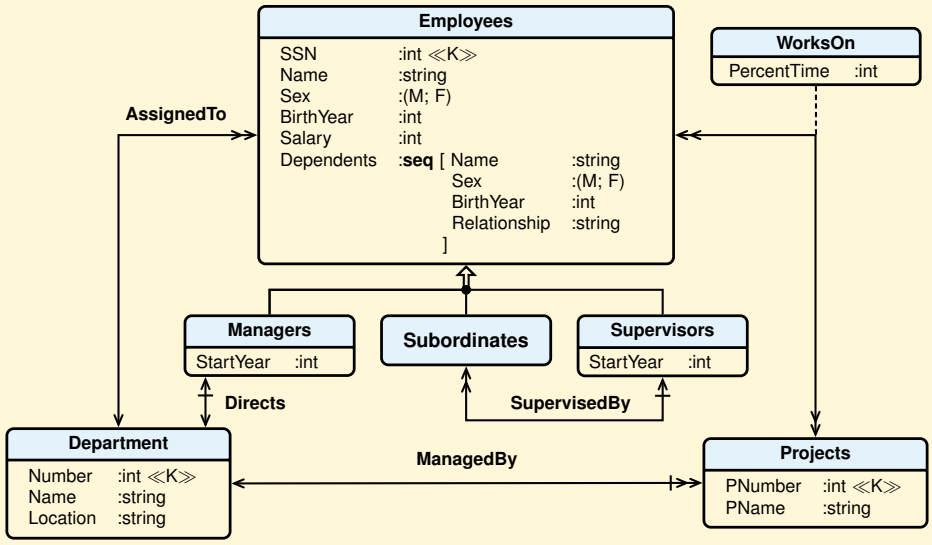
5. Design a conceptual schema for a Company database to keep track of a company's employees, departments, and projects.

   The company is organized into departments. Each department has a unique name, a unique number, a location, and a manager who is one of its employees. We keep track of the start year when the employee began managing the department.

   A department controls a number of projects, each of which has a name and a unique number.

   An employee has a name, a unique social security number, an address, a salary, a sex (m or f), and a birth year. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the percent-time that an employee works on each project. We also keep track of the direct supervisor of each employee, who belong to the same department, and the start year when the employee began acting as supervisor. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's name, sex, birth year, and relationship (spouse or child or other) to the employee (assume that only one parent works for the company). We are not interested in information about dependents once the parent leaves the company.

Solution



Chapter 3

THE RELATIONAL DATA MODEL

Section 3.2 Relational Algebra

3.2.4 Exercises

1. Prove the following properties:

   a) $\sigma_{\phi \wedge \psi}(E) = \sigma_{\phi}(E) \cap \sigma_{\psi}(E)$;

Solution

We prove each property $E_1 = E_2$ by showing how, for each ennuple t, you have $t \in E_1 \Leftrightarrow t \in E_2$. We take advantage of the following properties, which derive immediately from definition of relational operators.

$t \in \sigma_{\phi}(E) \Leftrightarrow t \in E \wedge \phi(t)$

$t \in \pi_{X}(E) \Leftrightarrow \exists t' \, (t' \in E \wedge t'[X] = t)$

$t \in (E_1 \overset{\bowtie}{_{\phi}} E_2) \Leftrightarrow \exists t', t'' \, (t' \in E_1 \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi(t))$

$t \in {}_{X}\gamma_{f_1(B_1)\ \mathbf{AS}\ C_1,\ \ldots,\ f_n(B_n)\ \mathbf{AS}\ C_n}(E)$

$$\Leftrightarrow \exists t' \, (\, t' \in E \wedge t = t'[X] \circ [C_1 = f_1(\{s[B_1] \mid s \in E \wedge s[X] = t'[X]\}) \circ \ldots \circ [C_n = f_n(\{s[B_n] \mid s \in E \wedge s[X] = t'[X]\})])$$

$\sigma_{\phi \wedge \psi}(E) = \sigma_{\phi}(E) \cap \sigma_{\psi}(E)$.

$$\begin{aligned} & t \in \sigma_{\phi \wedge \psi}(E) \\ \Leftrightarrow\ & t \in E \wedge \phi(t) \wedge \psi(t) \\ \Leftrightarrow\ & (t \in E \wedge \phi(t)) \wedge (t \in E \wedge \psi(t)) \\ \Leftrightarrow\ & (t \in \sigma_{\phi}(E)) \wedge (t \in \sigma_{\psi}(E)) \\ \Leftrightarrow\ & t \in (\sigma_{\phi}(E) \cap \sigma_{\psi}(E)) \end{aligned}$$

b) $\pi_X(\sigma_\phi(E)) = \sigma_\phi(\pi_X(E))$, if ϕ uses only attributes in X;

Solution

$\pi_X(\sigma_\phi(E)) = \sigma_\phi(\pi_X(E))$, if ϕ uses only attributes in X;

$t \in \pi_X(\sigma_\phi(E))$
$\Leftrightarrow\ \exists t'\ (t' \in \sigma_\phi(E) \wedge t'[X] = t)$
$\Leftrightarrow\ \exists t'\ (t' \in E \wedge \phi(t') \wedge t'[X] = t)$
$\Leftrightarrow$ (since ϕ only uses attributes in X)
$\exists t'\ (t' \in E \wedge t'[X] = t) \wedge \phi(t)$
$\Leftrightarrow\ t \in \pi_X(E) \wedge \phi(t)$
$\Leftrightarrow\ t \in \sigma_\phi(\pi_X(E))$

c) $\sigma_\phi(E_1 \underset{\phi_J}{\bowtie} E_2) = \sigma_\phi(E_1) \underset{\phi_J}{\bowtie} E_2$, if ϕ uses only attributes of E_1;

Solution

$\sigma_\phi(E_1 \underset{\phi_J}{\bowtie} E_2) = \sigma_\phi(E_1) \underset{\phi_J}{\bowtie} E_2$, if ϕ uses only attributes of E_1;

$t \in \sigma_\phi(E_1 \underset{\phi_J}{\bowtie} E_2)$
$\Leftrightarrow\ t \in (E_1 \underset{\phi_J}{\bowtie} E_2) \wedge \phi(t)$
$\Leftrightarrow\ \exists t', t''\ (t' \in E_1 \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi_J(t)) \wedge \phi(t)$
$\Leftrightarrow$ (since ϕ only uses attributes of E_1)
$\exists t', t''\ (t' \in E_1 \wedge \phi(t') \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi_J(t))$
$\Leftrightarrow\ \exists t', t''\ (t' \in \sigma_\phi(E_1) \wedge t'' \in E_2 \wedge t' \circ t'' = t \wedge \phi_J(t))$
$\Leftrightarrow\ t \in \sigma_\phi(E_1) \underset{\phi_J}{\bowtie} E_2$

d) $\sigma_\phi({}_A\gamma_F(E)) = {}_A\gamma_F(\sigma_\phi(E))$, if ϕ uses only attributes in A.

Solution

We use $[A = v]$ to denote an ennuple with a single attribute A with value v.
Let's assume, for simplicity, that F contains only one expression $f(B)$ **AS** C, and we prove:

$\sigma_\phi({}_A\gamma_F(E)) = {}_A\gamma_F(\sigma_\phi(E))$, if ϕ uses only attributes in A.

$t \in \sigma_\phi({}_A\gamma_F(E))$
$\Leftrightarrow\ t \in ({}_A\gamma_F(E)) \wedge \phi(t)$
$\Leftrightarrow\ \exists t'\ (t' \in E$
$\quad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in E \wedge s[A] = t'[A]\})]) \wedge \phi(t)$
$\Leftrightarrow\ \exists t'\ (t' \in E \wedge \phi(t)$

$$
\begin{aligned}
&\quad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in E \wedge s[A] = t'[A]\})])\\
\Leftrightarrow\ & \text{(since } \phi \text{ only uses attributes in } A \text{, and } t[A] = t'[A])\\
&\exists t'\ (t' \in E \wedge \phi(t')\\
&\quad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in E \wedge \phi(s) \wedge s[A] = t'[A]\})])\\
\Leftrightarrow\ & \exists t'\ (t' \in \sigma_{\phi}(E)\\
&\quad \wedge t = t'[A] \circ [C = f(\{s[B] \mid s \in \sigma_{\phi}(E) \wedge s[A] = t'[A]\})])\\
\Leftrightarrow\ & t \in {}_{A}\gamma_{F}(\sigma_{\phi}(E))
\end{aligned}
$$

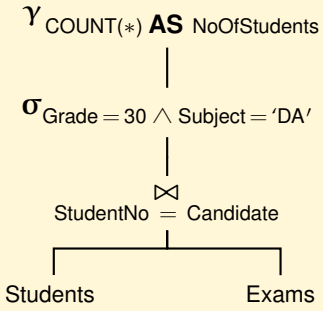
2. Consider the following database schema (the attributes of the primary key are underlined and those of the foreign key are marked with an asterisk)

   Students(StudentNo, Name, City, BirthYear)
   Exams(Subject, Candidate*, Grade, Date)

   Write the logical query plan for the following queries:

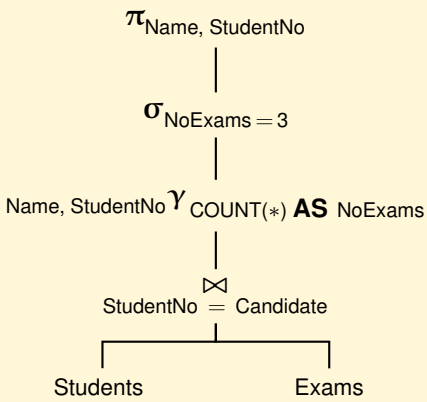
   a) Find the number of students who have passed the DA exam with grade 30.

Solution



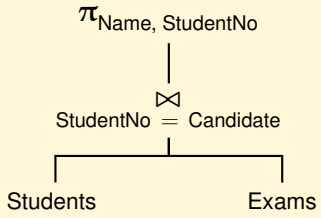
b) Find the name and the student number of students who have passed 3 exams.

Solution



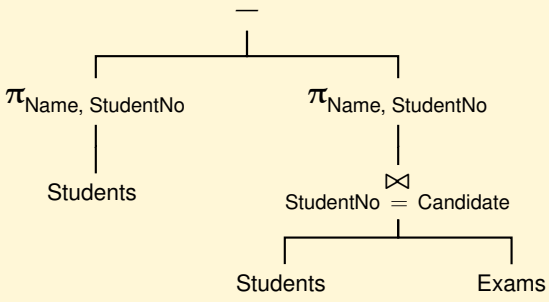
c) Find the name and the student number of students who have passed some exam.

Solution



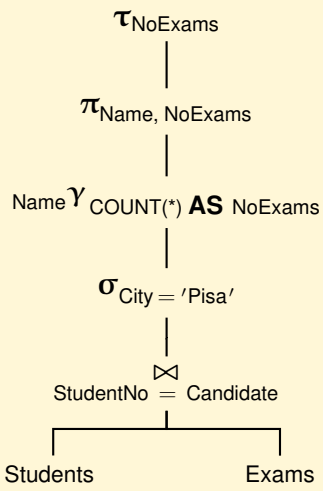
d) Find the name and the student number of students who have not passed some exam.

Solution



e) Find the names of the students of Pisa who have done some exam and the number of exams taken, sorted by number of exams.

Solution



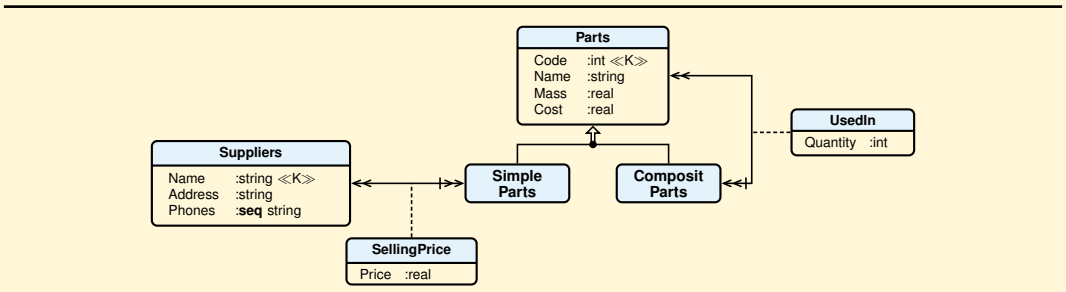
Section 3.3 Relational Database Design: ODM-to-Relational Mapping

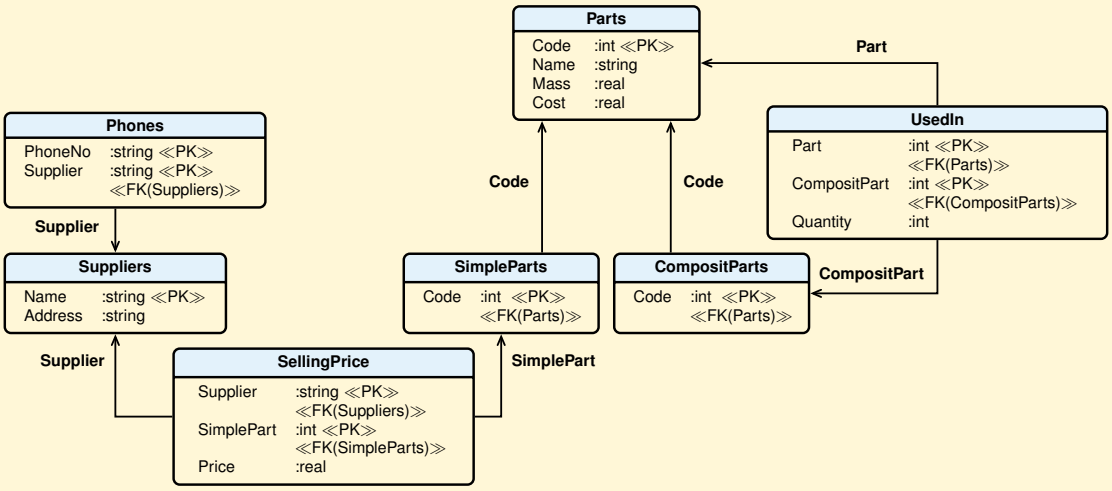
3.3.1 Exercise

Convert the following conceptual schemas to a relational database schema.

1. A solution to Exercise 2.4.7(3)

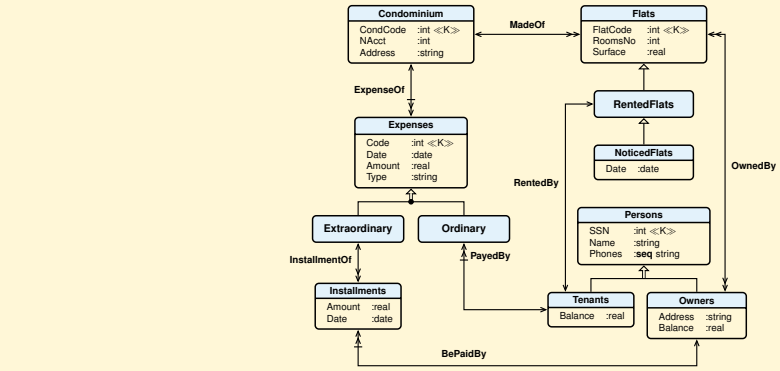
Solution

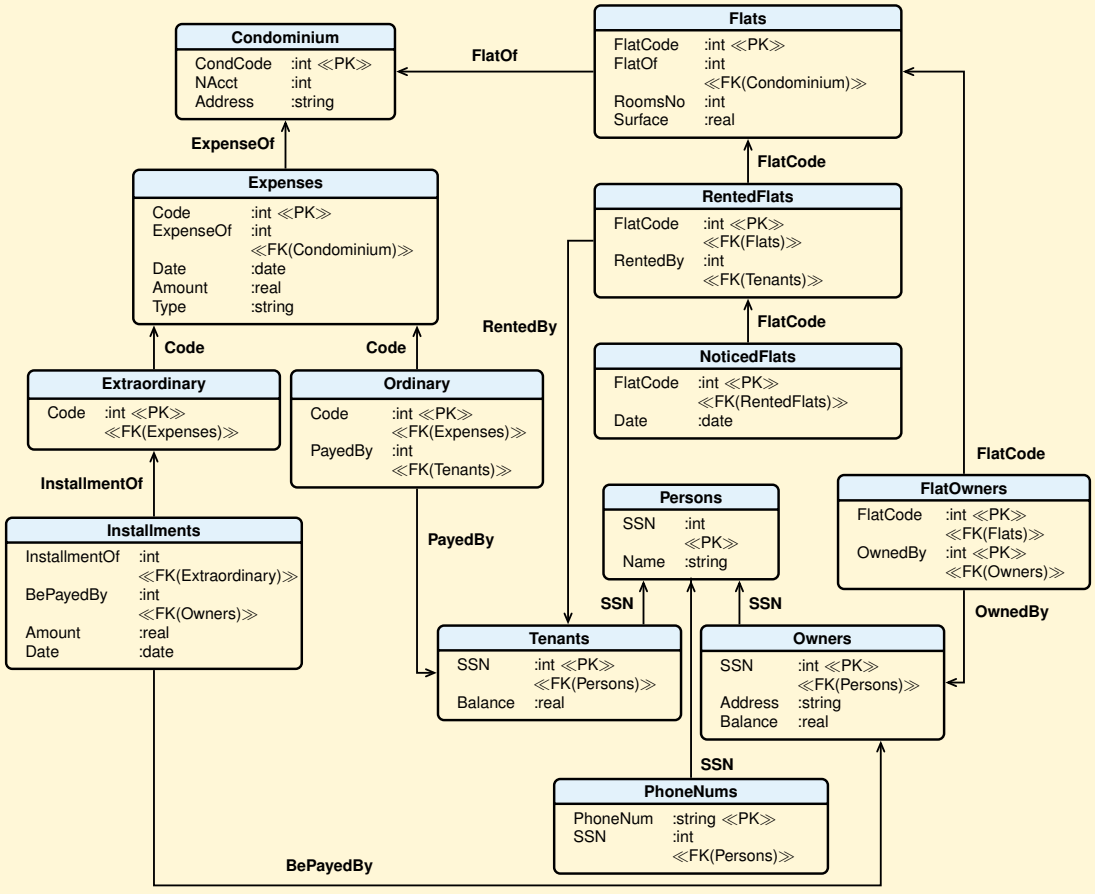




2. A solution to Exercise 2.4.7(4)

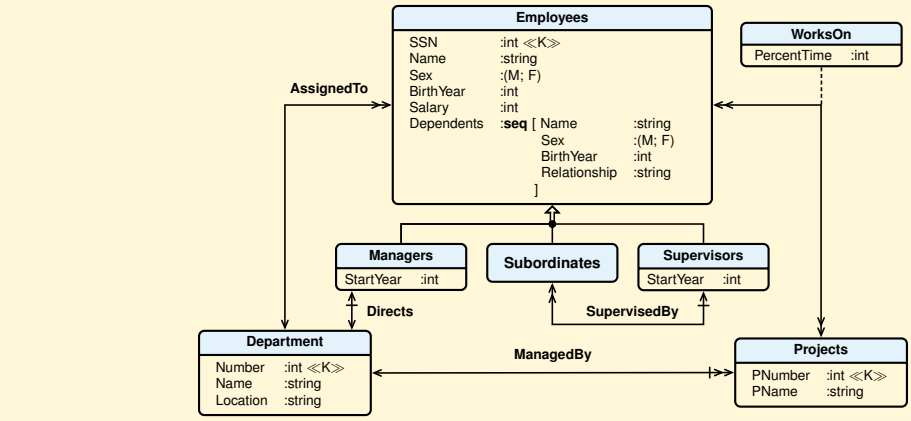
Solution

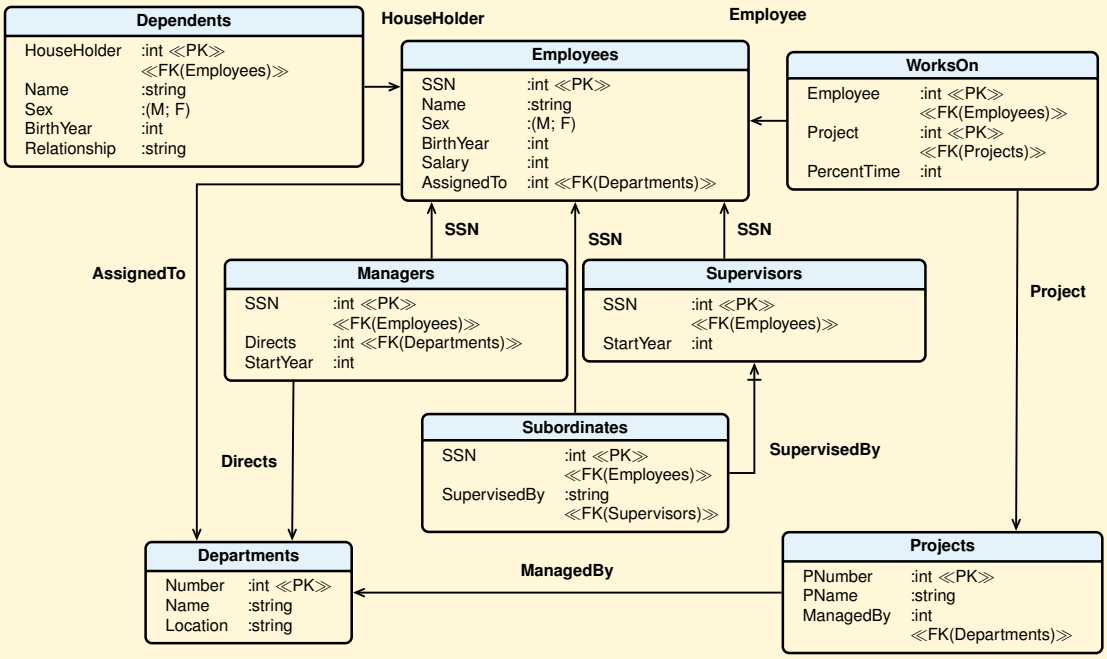




3. A solution to Exercise 2.4.7(5)

Solution





Section 3.4 Relational Database Design: Normalization Theory

3.4.10 Exercises

1. Prove that for a schema $R\langle T, F\rangle$, with F a *canonical cover*, if an attribute A_i does not appear on the right side of any FD, then A_i belongs to every key of R.

> **Solution**
> Consider any Y such that $A \notin Y$. Looking at the operation of the slow closing algorithm, we observe that A cannot belong to Y^+, so $A \notin Y$ implies that Y cannot be superkey, and so Y can't even be a key.

2. Prove that if a schema $R\langle T, F\rangle$ has two attributes AB only, then it is in BCNF.

> **Solution**
> The only possible (non-trivial) FD's are $A \to B$ and $B \to A$. So, there are four possible cases:
>
> – No FD's holds in R: the key is AB and R is in BCNF.
> – Only $A \to B$ holds: the key is A and R is in BCNF.
> – Only $B \to A$ holds: the key is B and R is in BCNF.
> – Both $A \to B$ and $B \to A$ holds: A and B are keys and R is in BCNF.
>
> Hence, every relation with two attributes is always in BCNF!

3. Prove that if a schema $R\langle T, F\rangle$ is in 3NF, and all keys are made of one attributes, then it is in BCNF. *Hint*: prove that for each $X \to A \in F$, X is a superkey.

> **Solution**
> Let us prove that for each $X \to A \in F$, X is a superkey. Suppose by contradiction that $X \to A$ and X is not superkey, then A is *prime* for the 3NF, and it is also a key: $A \to T$. From $X \to A$ and $A \to T$ it follows that $X \to T$ and therefore it is superkey (contradiction).

4. For each of the following relational schemas $R\langle T, F\rangle$, with F a *canonical cover*:
   (a) $R\langle\{A, B, C, D\}, \{A \to B, A \to C\}\rangle$
   (b) $R\langle\{A, B, C, D\}, \{AB \to C, C \to D, D \to A\}\rangle$
   (c) $R\langle\{A, B, C, D, E, F\}, \{A \to C, DE \to F, B \to D\}\rangle$
   (d) $R\langle\{A, B, C, D, E\}, \{AD \to B, CB \to A, DE \to A, A \to E\}\rangle$

do the following:

(a) Find all the keys of R,
(b) Indicate all the BCNF violations.
(c) Decompose the relation, as necessary, into collections of relations that are in BCNF. Say if the decomposition is dependency preserving.
(d) Indicate all the 3NF violations.
(e) If the relation is not in 3FN, decompose it into collections of relations that are in 3NF and are data preserving.

Solution

1. $R\langle\{A, B, C, D\}, \{A \to B, A \to C\}\rangle$

   (a) *Find all the keys of* R.

   The attributes A and D never appear on the right-hand sides of the given FDs, so they must be present in all the possibile keys.
   Since $AD^+ = ABCD$ cover all the relation attributes, AD is the only key.

   (b) *Indicate all the BCNF violations.*

   $A \to B$ and $A \to C$ violate BCNF because the determinants are not keys.

   (c) *Decompose the relation, as necessary, into collections of relations that are in BCNF. Say if the decomposition is dependency preserving.*

   Decomposition based on $A \to B$:
   - $R\langle\{A, B, C, D\}, \{A \to B, A \to C\}\rangle$
   - $R1\langle\{A, B\}, \{A \to B\}\rangle, R2\langle\{A, C, D\}, \{A \to C\}\rangle$
   - $R1\langle\{A, B\}, \{A \to B\}\rangle, R2\langle\{A, C\}, \{A \to C\}\rangle, R3\langle\{A, D\}, \{\}\rangle$.

   The dependencies are preserved.

   Decomposition based on $A \to C$:
   - $R\langle\{A, B, C, D\}, \{A \to B, A \to C\}\rangle$
   - $R1\langle\{A, C\}, \{A \to C\}\rangle, R2\langle\{A, B, D\}, \{A \to B\}\rangle$
   - $R1\langle\{A, C\), \{A \to C\}\rangle, R2\langle\{A, B\)\}, \{A \to B\}\rangle, R3\langle\{A, D\), \{\}\rangle$

   The dependencies are preserved.

   (d) *Indicate all the 3NF violations.*

   The schema is not in 3NF because in all the functional dependencies $X \to H \in F$, X is not a key and H is not prime.

   (e) *If the relation is not in 3FN, decompose it into collections of relations that are in 3NF and are data preserving.*

   The decomposition is:

   $R1\langle\{A, B, C\}, \{A \to B, C\}\rangle, R2\langle\{A, D\}, \{\}\rangle$

2. $R\langle\{A, B, C, D\}, \{AB \to C, C \to D, D \to A\}\rangle$

   (a) *Find all the keys of* R.

   The attribute B never appear on the right-hand sides of the given FDs, and so it must be present in all the possibile keys.
   $B^+ = B$ is not a key because does not cover all the relation attributes. Considering for addition to B the attributes A C D, the possible keys are AB, BC, BD.

   (b) *Indicate all the BCNF violations.*

   $C \to D$ and $D \to A$ violates BCNF because the determinants are not keys.

   (c) *Decompose the relation, as necessary, into collections of relations that are in BCNF. Say if the decomposition is dependency preserving.*

   Decomposition based on $C \to D$:
   - $R\langle\{A, B, C, D\}, \{AB \to C, C \to D, D \to A\}\rangle$
   - $R1\langle\{C, D\}, \{C \to D\}\rangle, R2\langle\{A, B, C\}, \{AB \to C\}\rangle$

   The decomposition is not dependency preserving ($D \to A$ has been lost).

   Decomposition based on $D \to A$:
   - $R\langle\{A, B, C, D\}, \{AB \to C, C \to D, D \to A\}\rangle$
   - $R1\langle\{A, D\}, \{D \to A\}\rangle, R2\langle\{B, C, D\}, \{C \to D\}\rangle$
   - $R1\langle\{A, D\}, \{D \to A\}\rangle, R3\langle\{C, D\}, \{C \to D\}\rangle, R4\langle\{B, C\}, \{\}\rangle$

   The decomposition is not dependency preserving ($AB \to C$ has been lost).

   (d) *Indicate all the 3NF violations.*

   The schema is in 3NF because all the determined attributes in F are primes.

3. $R\langle\{A, B, C, D, E, F\}, \{A \to C, DE \to F, B \to D\}\rangle$

   (a) *Find all the keys of* R.

   Since the attributes A, B and E never appear on the right-hand sides of the given FD, then they must be included in all the possibile keys.
   Since $ABE^+ = ABCDEF$ covers all the relation attributes, ABE is the only key.

   (b) *Indicate all the BCNF violations.*

   $A \to C$, $B \to D$ and $DE \to F$ violate BCNF because the determinants are not keys.

   (c) *Decompose the relation, as necessary, into collections of relations that are in BCNF. Say if the decomposition is dependency preserving.*

   Decomposition based on $A \to C$:

   - $R\langle\{A, B, C, D, E, F\}, \{A \to C, DE \to F, B \to D\}\rangle$
   - $R1\langle\{A, C\}, \{A \to C\}\rangle, R2\langle\{A, B, D, E, F\}, \{DE \to F, B \to D\}\rangle$
   - $R1\langle\{A, C\}, \{A \to C\}\rangle, R3\langle\{D, E, F\}, \{DE \to F\}\rangle, R4\langle\{A, B, D, E\}, \{B \to D\}\rangle$
   - $R1\langle\{A, C\}, \{A \to C\}\rangle, R3\langle\{D, E, F\}, \{DE \to F\}\rangle, R5\langle\{B, D\}, \{B \to D\}\rangle,$ $R6\langle\{A, B, E\}, \{\}\rangle$

   The decomposition is dependency preserving.

   Another possible decomposition based on $A \to C$:

   - $R\langle\{A, B, C, D, E, F\}, \{A \to C, DE \to F, B \to D\}\rangle$
   - $R1\langle\{A, C\}, \{A \to C\}\rangle, R2\langle\{A, B, D, E, F\}, \{DE \to F, B \to D\}\rangle$
   - $R1\langle\{A, C\}, \{A \to C\}\rangle, R3\langle\{B, D\}, \{B \to D\}\rangle, R4\langle\{A, B, E, F\}, \{BE \to F\}\rangle$
   - $R1\langle\{A, C\}, \{A \to C\}\rangle, R3\langle\{B, D\}, \{B \to D\}\rangle, R5\langle\{B, E, F\}, \{BE \to F\}\rangle,$ $R6\langle\{A, B, E\}, \{\}\rangle$

   The decomposition is not dependency preserving ($DE \to F$ has been lost).

   (d) *Indicate all the 3NF violations.*

   The schema is not in 3NF because in all the functional dependencies $X \to H \in F$, X is not a key and H is not prime.

   (e) *If the relation is not in 3FN, decompose it into collections of relations that are in 3NF and are data preserving.*

   - from $A \to C$ follows $R1(A, C)$
   - from $DE \to F$ follows $R2(D, E, F)$
   - from $B \to D$ follows $R3(B, D)$
   - and $R4(A, B, E)$ is added to have a decomposition data and dependency preserving.

4. $R\langle\{A, B, C, D, E\}, \{AD \to B, CB \to A, DE \to A, A \to E\}\rangle$

   (a) *Find all the keys of* R.

   Since the attributes C and D never appear on the right-hand sides of the given FD, then they must be included in all the possibile keys.
   Since $CD^+ = CD$ is not a key because does not cover all the relation attributes, the possible keys are ACD, BCD, CDE.

   (b) *Indicate all the BCNF violations.*

   AD $\to$ B, CB $\to$ A, DE $\to$ A, A $\to$ E violate BCNF because the determinants are not keys.

   (c) *Decompose the relation, as necessary, into collections of relations that are in BCNF. Say if the decomposition is dependency preserving.*

   Decomposition based on AD $\to$ B:

   - $R\langle\{A, B, C, D, E\}, \{AD \to B, CB \to A, DE \to A, A \to E\}\rangle$
   - $R1\langle\{A, B, D\}, \{AD \to B\}\rangle$, $R2\langle\{A, C, D, E\}, \{DE \to A, A \to E\}\rangle$
   - $R1\langle\{A, B, D\}, \{AD \to B\}\rangle$, $R3\langle\{A, E\}, \{A \to E\}\rangle$, $R4\langle\{A, C, D\}, \{\}\rangle$

   The dependencies DE $\to$ A, BC $\to$ A are not preserved in the decomposition.

   Decomposition based on A $\to$ E:

   - $R\langle\{A, B, C, D, E\}, \{AD \to B, CB \to A, DE \to A, A \to E\}\rangle$
   - $R1\langle\{A, E\}, \{A \to E\}\rangle$, $R2\langle\{A, B, C, D\}, \{AD \to B, CB \to A\}\rangle$
   - $R1\langle\{A, E\}, \{A \to E\}\rangle$, $R3\langle\{A, B, C\}, \{CB \to A\}\rangle$, $R4\langle\{B, C, D\}, \{\}\rangle$

   The dependencies DE $\to$ A, AD $\to$ B are not preserved in the decomposition.

   (d) *Indicate all the 3NF violations.*

   The schema is in 3NF because all the determined attributes in F are primes.

Chapter 5

SQL: A RELATIONAL DATABASE LANGUAGE

5.10 Exercises

Give a relational schema in SQL for your solution to Exercise 3.2(3), and write the following queries:

1. Retrieve the name and birth year of the employee's child with code 350.
2. For each employee, retrieve the employee name and the name of the department where he works.
3. Retrieve the names and birth years of female employees older than their supervisor.
4. Retrieve the names of employees who do not have supervisors.
5. Retrieve the names of employees who work for the Research department, and the location of the department.
6. For every project located in Pisa, list the project number and name, the controlling department name, and the department manager's name.
7. Retrieve the names of employees who have no dependents.
8. Retrieve the names of supervisors who have at least one dependent.
9. For each employee retrieve the employees name and the name of the immediate supervisor.
10. Retrieve the names of employees who have a dependent with the same sex.
11. For each employee retrieve the employees name and the social security number, the name of the project on which he works, and the name of the department that manages the project, sorted by the names of the department and the employee.
12. For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.
13. For each project, retrieve the project number, the project name and the number of employees from department 10 who work on the project.
14. Retrieve the names of employees who have all dependants with their same sex.
15. Retrieve only the names of employees who have all dependants with the same sex.
16. Retrieve the names of employees who only work on projects for 20 percent-time.
17. Find the name of the employees who work at least on all the projects in which the employee with code 100 participates.

Solution

A relational schema in the JRS SQL.

```
CREATE DATABASE Company IN CompanyBD;

CREATE TABLE Departments (
    Number          INTEGER NOT NULL,
    Name            VARCHAR (16) NOT NULL,
    Location        VARCHAR (16) NOT NULL,
PRIMARY KEY         (Number),
UNIQUE              (Name, Location)    );


CREATE TABLE Employees (
    SSN             INTEGER NOT NULL,
    Name            VARCHAR (20) NOT NULL,
    Sex             VARCHAR(1) NOT NULL,
    BirthYear       INTEGER NOT NULL,
    Salary          INTEGER NOT NULL,
    AssignedTo      INTEGER NOT NULL,
PRIMARY KEY         (SSN),
FOREIGN KEY         (AssignedTo)
REFERENCES          Departments ON DELETE CASCADE  );


CREATE TABLE Dependents (
    HouseHolder     INTEGER NOT NULL,
    Name            VARCHAR(16) NOT NULL,
    Sex             VARCHAR(1) NOT NULL,
    BirthYear       INTEGER NOT NULL,
    Relationship    VARCHAR(8) NOT NULL,
PRIMARY KEY         (HouseHolder, Name),
FOREIGN KEY         (HouseHolder)
REFERENCES          Employees ON DELETE CASCADE );


CREATE TABLE Supervisors (
    SSN             INTEGER NOT NULL,
    StartYear       INTEGER NOT NULL,
PRIMARY KEY         (SSN),
FOREIGN KEY         (SSN)
REFERENCES          Employees ON DELETE CASCADE );
```

```
CREATE TABLE Subordinates (
    SSN             INTEGER NOT NULL,
    SupervisedBy    INTEGER,
PRIMARY KEY         (SSN),
FOREIGN KEY         (SSN)
REFERENCES          Employees ON DELETE CASCADE,
FOREIGN KEY         (SupervisedBy)
REFERENCES          Supervisors ON DELETE SET NULL );

CREATE TABLE Managers (
    SSN             INTEGER NOT NULL,
    Directs         INTEGER NOT NULL,
    StartYear       INTEGER NOT NULL,
PRIMARY KEY         (SSN),
FOREIGN KEY         (SSN)
REFERENCES          Employees ON DELETE CASCADE,
FOREIGN KEY         (Directs)
REFERENCES          Departments ON DELETE CASCADE );

CREATE TABLE Projects (
    PNumber         INTEGER NOT NULL,
    PName           VARCHAR (10) NOT NULL,
    ManagedBy       INTEGER NOT NULL,
PRIMARY KEY         (PNumber),
FOREIGN KEY         (ManagedBy)
REFERENCES          Departments ON DELETE CASCADE );

CREATE TABLE WorksOn (
    PercentTime     INTEGER NOT NULL,
    Project         INTEGER NOT NULL,
    Employee        INTEGER NOT NULL,
PRIMARY KEY         (Employee, Project) ,
FOREIGN KEY         (Employee)
REFERENCES          Employees ON DELETE CASCADE,
FOREIGN KEY         (Project)
REFERENCES          Projects ON DELETE CASCADE );
```

All queries assume the existence of the following views:

```
CREATE VIEW DataSupervisors (
    SSN, Name, Sex, BirthYear, Salary, AssignedTo, StartYear) AS
SELECT   e.SSN, e.Name, e.Sex, e.BirthYear,
         e.Salary, e.AssignedTo, s.StartYear
FROM     Employees e, Supervisors s
WHERE    e.SSN = s.SSN;


CREATE VIEW DataManagers (
    SSN, Name, Sex, BirthYear, Salary, AssignedTo, Directs, StartYear) AS
SELECT   e.SSN, e.Name, e.Sex, e.BirthYear,
         e.Salary, e.AssignedTo, m.Directs, m.StartYear
FROM     Employees e, Managers m
WHERE    e.SSN = m.SSN;


CREATE VIEW DataSubordinates (
    (SSN, Name, Sex, BirthYear, Salary, AssignedTo, SupervisedBy) AS
SELECT   e.SSN, e.Name, e.Sex, e.BirthYear,
         e.Salary, e.AssignedTo, s.SupervisedBy
FROM     Employees e, Subordinates s
WHERE    e.SSN = s.SSN;
```

SQL queries

1. Retrieve the name and birth year of the employee's child with code 350.

   ```
   SELECT   Name, BirthYear
   FROM     Dependents
   WHERE    HouseHolder = 350 AND Relationship = 'child';
   ```

2. For each employee, retrieve the employee name and the name of the department where he works.

   ```
   SELECT   Name, D.Name
   FROM     Employees E, Departments D
   WHERE    E.AssignedTo = D.Number;
   ```

3. Retrieve the names and birth years of female employees older than their supervisor.

```
SELECT   U.Name, U.BirthYear
FROM     DataSubordinates U
WHERE    U.Sex = 'f'
         AND U.BirthYear > (
         SELECT   SE.BirthYear
         FROM     DataSupervisors SE
         WHERE    U.SupervisedBy = SE.SSN );
```

4. Retrieve the names of employees who do not have supervisors.

```
SELECT   Name
FROM     DataSubordinates U
WHERE    U.SupervisedBy IS NULL
   UNION
SELECT   Name
FROM     DataManagers
   UNION
SELECT   Name
FROM     DataSupervisors;
```

5. Retrieve the names of employees who work for the Research department, and the department location.

```
SELECT   E.Name, D.Location
FROM     Employees E, Departments D
WHERE    E.AssignedTo = D.Number AND D.Name = 'Research';
```

6. For every project located in 'Pisa', list the project number and name, the controlling department name, and the department manager's name.

```
SELECT   P.PNumber, P.PName, D.Name, M.Name
FROM     Projects P, DataManagers M, Departments D
WHERE    P.PNumber = D.Number AND M.Manages = D.Number
         AND D.Location = 'Pisa';
```

7. Retrieve the names of employees who have no dependents.

```
SELECT   E.Name
FROM     Employees E
WHERE    NOT EXISTS(
         SELECT *
         FROM    Dependents D
         WHERE  D.HouseHolder = E.SSN);
```

or:

```
SELECT   Name
FROM     Employees
  EXCEPT
SELECT   Name
FROM     Employee, Dependents
WHERE    SSN = HouseHolder ;
```

8. Retrieve the names of supervisors who have at least one dependent.

```
SELECT   E.Name
FROM     Employees E, Supervisors S
WHERE    E.SSN = S.SSN
         AND EXISTS (
         SELECT    *
         FROM      Dependents D
         WHERE     E.SSN = D.SSN );
```

9. For each employee retrieve the employees name and the name of the immediate supervisor.

```
SELECT   E.Name, SE.Name
FROM     Employees E, Supervisors S, Employees SE, Subordinates U
WHERE    E.SSN = U.SSN AND S.SSN = SE.SSN AND U.SupervisedBy = S.SSN;
```

10. Retrieve the names of employees who have a dependent with the same sex.

```
SELECT   E.Name
FROM     Employees E, Dependents D
WHERE    E.SSN = D.SSN AND E.Sex = D.Sex;
```

11. For each employee retrieve the employees name and the social security number, the name of the project on which he works, and the name of the department that manages the project, sorted by the names of the department and the employee.

```
SELECT      E.SSN, E.Name, P.PName, D.Name
FROM        Employees E, WorksOn W, Departments D, Projects P
WHERE       E.SSN = W.Employee AND P.PNumber = W.Project
            AND P.DNumber=D.DNumber
ORDER BY    D.Name, E.Name;
```

12. For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT      P.PNumber, P.PName, COUNT(*)
FROM        Projects P, WorksOn W, Employees E
WHERE       P.PNumber = W.Project AND W.Employee = E.SSN
GROUP BY    P.PNumber, P.PName
HAVING      COUNT(*) > 2;
```

13. For each project, retrieve the project number, the project name and the number of employees from department 10 who work on the project.

```
SELECT      P.PNumber, P.PName, COUNT(*)
FROM        Projects P, WorksOn W, Employees E
WHERE       P.PNumber = W.Project AND W.Employee = E.SSN AND P.PNumber = 10
GROUP BY    P.PNumber, P.PName ;
```

14. Retrieve the names of employees who have all dependants with their same sex.

    Let us write the query using *universal quantification*:

```
SELECT   E.Name
FROM     Employee E
WHERE    (FOR ALL D IN Dependants WHERE D.HouseHolder = E.SSN:
                  D.Sex = E.Sex)
         AND (FOR SOME D IN Dependants WHERE D.HouseHolder = E.SSN:
                  TRUE);
```

    Let us rewrite the query using *existential quantification*:

```
SELECT   E.Name
FROM     Employee E
WHERE    (NOT FOR SOME D IN Dependants WHERE D.HouseHolder = E.SSN:
                  NOT (D.Sex = E.Sex) )
         AND (FOR SOME D IN Dependants WHERE D.HouseHolder = E.SSN:
                  TRUE);
```

    Hence the SQL query is:

```
SELECT   E.Name
FROM     Employee E
WHERE    (NOT EXISTS ( SELECT *
                       FROM   Dependants D
                       WHERE  D.HouseHolder = E.SSN
                              AND NOT (D.Sex = E.Sex) )
AND      EXISTS ( SELECT *
                  FROM   Dependants D
                  WHERE  D.HouseHolder = E.SSN);
```

The query can also be written appropriately using the join and aggregation functions:

```
SELECT     E.Name
FROM       Employee E, Dependants D
WHERE      D.HouseHolder = E.SSN
AND        EXISTS ( SELECT *
                    FROM   Dependants F
                    WHERE  F.HouseHolder = E.SSN
                           AND D.Sex = E.Sex)
GROUP BY   E.SSN, E.Name
HAVING     COUNT(DISTINCT D.Sex) = 1;
```

15. Retrieve only the names of employees who have all dependants with the same sex.

Let us write the query using *universal quantification*:

```
SELECT   E.Name
FROM     Employees E
WHERE    ( (FOR ALL D IN Dependants WHERE D.HouseHolder = E.SSN:
              D.Sex = 'm') OR
           (FOR ALL D IN Dependants WHERE D.HouseHolder = E.SSN:
              D.Sex = 'f') ) AND
           (FOR SOME D IN Dependants WHERE D.HouseHolder = E.SSN:
              TRUE);
```

Let us rewrite the query using *existential quantification*:

```
SELECT   E.Name
FROM     Employees E
WHERE    ( (NOT FOR SOME D IN Dependants WHERE D.HouseHolder = E.SSN:
              NOT (D.Sex = 'm') ) OR
           (NOT FOR SOME D IN Dependants WHERE D.HouseHolder = E.SSN:
              NOT (D.Sex = 'f')) ) AND
           (FOR SOME D IN Dependants WHERE D.HouseHolder = E.SSN:
              TRUE);
```

Hence the SQL query is:

```
SELECT  E.Name
FROM    Employees E
WHERE   ( NOT EXISTS ( SELECT *
                       FROM   Dependants D
                       WHERE  D.HouseHolder = E.SSN AND NOT (D.Sex = 'm'))
OR      NOT EXISTS (   SELECT *
                       FROM   Dependants D
                       WHERE  D.HouseHolder = E.SSN AND NOT (D.Sex = 'f')) )
AND           EXISTS (  SELECT *
                       FROM   Dependants D
                       WHERE  D.HouseHolder = E.SSN );
```

The query can also be written without subselects, appropriately using the join and aggregation functions:

```
SELECT     E.Name
FROM       Employees E, Dependants D
WHERE      D.HouseHolder = E.SSN
GROUP BY   E.SSN, E.Name
HAVING     COUNT(DISTINCT D.Sex) = 1;
```

16. Retrieve the names of employees who only work on project for 20 percent time.

```
SELECT  E.Name
FROM    Employees E
WHERE   NOT EXISTS (
              SELECT  *
              FROM    WorksOn W
              WHERE   W.Employee = E.SSN AND W.PercentTime <> 20)
        AND EXISTS (
              SELECT  *
              FROM    WorksOn W
              WHERE   W.Employee = E.SSN );
```

17. Retrieve the name of each employee who works at least on all the project to which employee 100 participates.

    Let us write the query using *universal quantification*:

```
SELECT  E.Name
FROM    Employees E
WHERE   FOR ALL W IN WorksOn WHERE W.Employee = 100:
           (FOR SOME V IN WorksOn WHERE V.Employee = E.SSN:
                 W.Project = V.Project );
```

    Let us rewrite the query using *existential quantification*:

```
SELECT  E.Name
FROM    Employees E
WHERE   NOT FOR SOME W IN WorksOn WHERE W.Employee = 100:
              (NOT FOR SOME V IN WorksOn WHERE V.Employee = E.SSN:
                    W.Project = V.Project );
```

Hence the SQL query is:

```
SELECT   E.Name
FROM     Employees E
WHERE    NOT EXISTS
         (SELECT *
          FROM    WorksOn W
          WHERE   W.Employee = 100
                  AND NOT EXISTS
                        (SELECT  *
                         FROM    WorksOn V
                         WHERE   V.Employee = E.SSN
                                 AND W.Project = V.Project ));
```

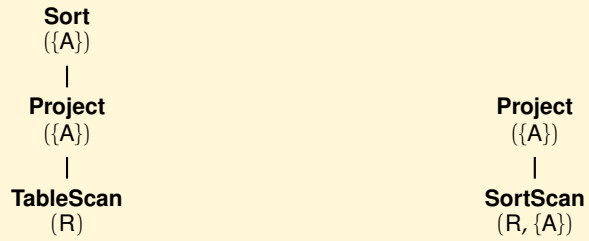
Chapter 6

DBMS ARCHITECTURE

6.8 Exercises

1. Say which of the following statements is true or false and justify the answer:

   1) The following physical plans for the query **SELECT** A **FROM** R **ORDER BY** A are not equivalent:



   2) Logical operator and physical operator are synonyms.
   3) Each SQL query has multiple logical trees.
   4) Each logical tree has multiple physical trees.
   5) Managing concurrency with data blocking does not create deadlocks.
   6) With the undo-redo method, no data modified by a transaction can be carried over to the BD before the corresponding log record is written in the *permanent memory.*
   7) With the redo method, all changes to a T must be reflected in the BD before the commit record is written in the log.

Solution

1) *False*. The first plan executes the query with an algorithm which first retrieves the R records, then projects them on A and finally sorts them. The second plan uses an algorithm that first orders R then retrieves the sorted records and finally projects them on A, producing the same result.

2) *False*. A logical operator is an operator of relational algebra, while a physical operator is an algorithm for implementing a logical operator using the storage structures available in the *storage engine*.

3) *True*. The initial logical tree can be transformed in different ways by applying the equivalence rules of relational algebra. .

4) *True*. A logical operator can be implemented with different algorithms.

5) *False*. The data blocking technique can create deadlocks.

6) *True*. The undo-redo method follows the rules for *undo* and *redo*: before modifying the BD, the old version and the new version of the modified data are written in the log.

7) *False*. The redo method does not require transaction changes to be reflected in the BD before the commit record is written in the log.

2. Give a logical and physical query plans for the following queries based on the database schema (the attributes of the primary key are underlined and those of the foreign key are marked with an asterisk).

   For a physical query plan consider two cases: without using indexes and using indexes that you believe are useful to speed-up the query.

   Departments(Number, Name, Location)
   Employees(SSN, Name, BirthYear, Sex, AssighedToDpt*)
   Managers(SSN, StartYear, DirectsDpt*)
   Projects(PNumber, PName, ManagedByDpt*)
   WorksOn(Project*, Employee*, TimeShare)

   1) **SELECT** **DISTINCT** E.SSN, E.Name, P.TimeShare
      FROM Employees E, WorksOn W
      WHERE E.SSN= W.Employee **AND** W.TimeShare $>$ 50
      ORDER BY E.SSN;

   2) **SELECT** E.SSN, E.Name, COUNT(*)
      FROM Employees E, WorksOn W
      WHERE E.SSN= W.Employee **AND** W.TimeShare $>$ 50
      GROUP BY E.SSN, E.Name;

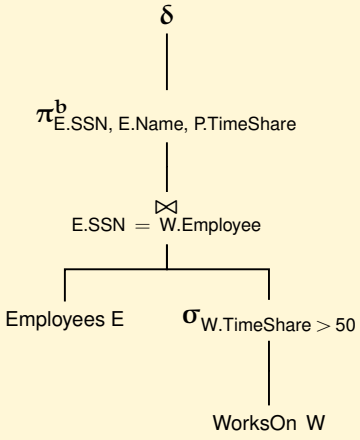
   3) **SELECT** E.Name, M.StartTime, D.Name, D.Location
      FROM Managers M, Employees E, Departments D
      WHERE M.SSN = E.SSN **AND** M.DirectsDpt = D.Number **AND** D.Location = 'Pisa'
      ORDER BY E.Name;

   4) **SELECT** E.SSN, E.Name, COUNT(*), SUM(W.TimeShare)
      FROM Employees E, WorksOn W
      WHERE E.SSN= W.Employee **AND** W.TimeShare $=$ 100
      GROUP BY E.SSN, E.Name
      HAVING COUNT(*) $>$ 2 **ORDER BY** E.Name;

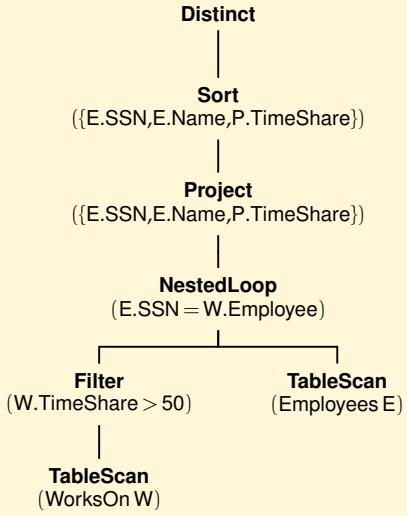
Solution

Let us assume that there is only an index IdxEPk on the Emplyees primary key SSN and to execute the join with the **IndexNestedLoop**:

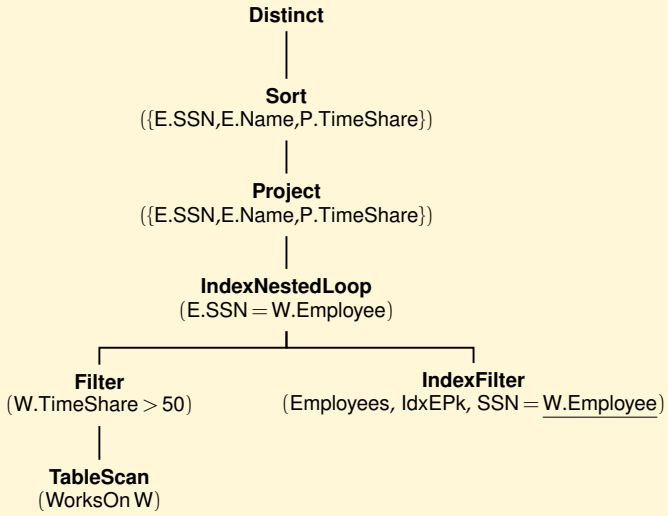
1) **SELECT** **DISTINCT** E.SSN, E.Name, P.TimeShare
FROM Employees E, WorksOn W
WHERE E.SSN= W.Employee **AND** W.TimeShare $>$ 50
ORDER BY E.SSN;



Logical query plan

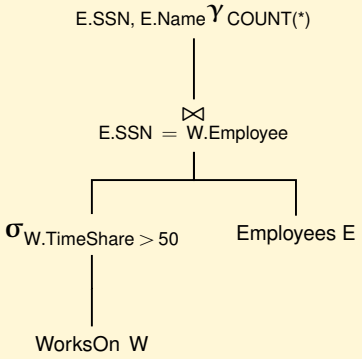


Physical query plan without indexes

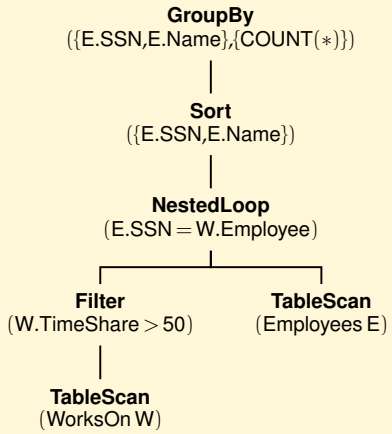


Physical query plan with indexes

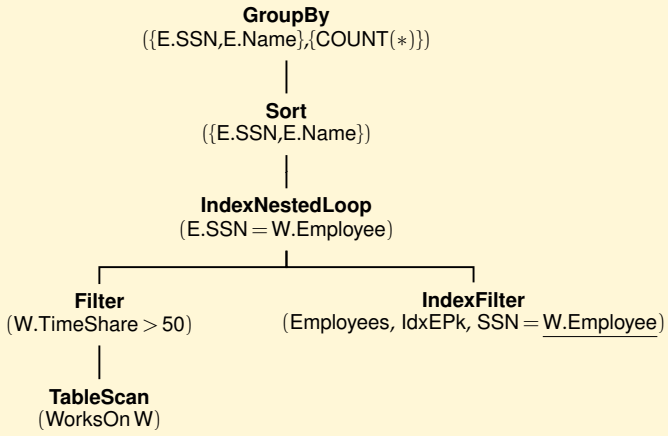
2) **SELECT** E.SSN, E.Name, COUNT(*)
FROM Employees E, WorksOn W
WHERE E.SSN= W.Employee **AND** W.TimeShare $>$ 50
GROUP BY E.SSN, E.Name;



Logical query plan

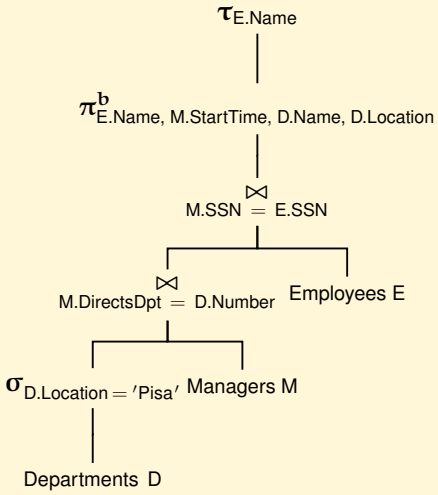


Physical query plan without indexes

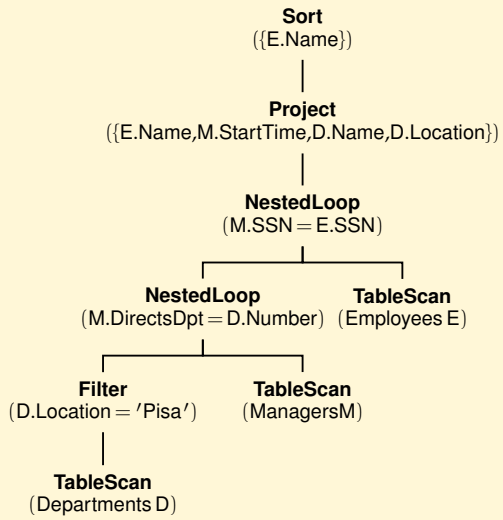


Physical query plan with indexes

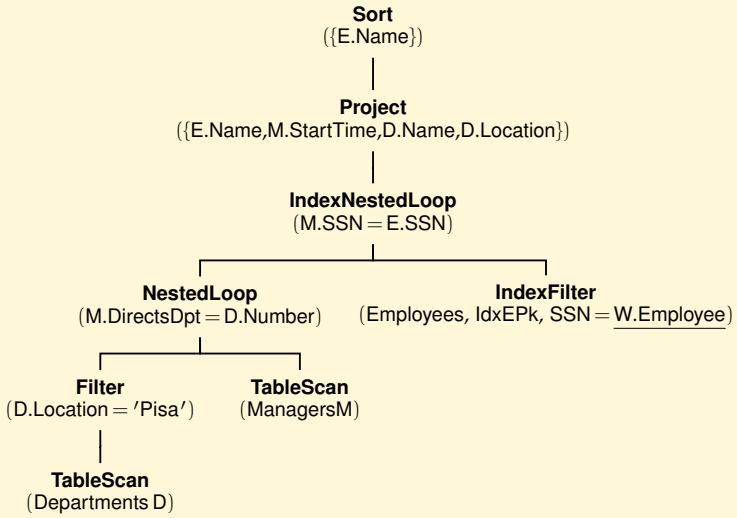
3) **SELECT** E.Name, M.StartTime, D.Name, D.Location
FROM Managers M, Employees E, Departments D
WHERE M.SSN = E.SSN **AND** M.DirectsDpt = D.Number **AND** D.Location = 'Pisa'
ORDER BY E.Name;



Logical query plan

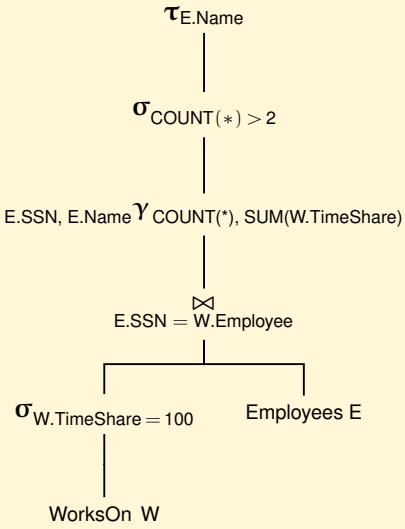


Physical query plan without indexes

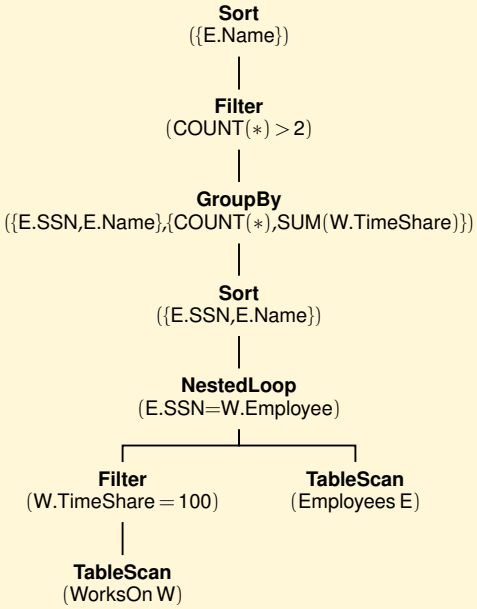


Physical query plan with indexes

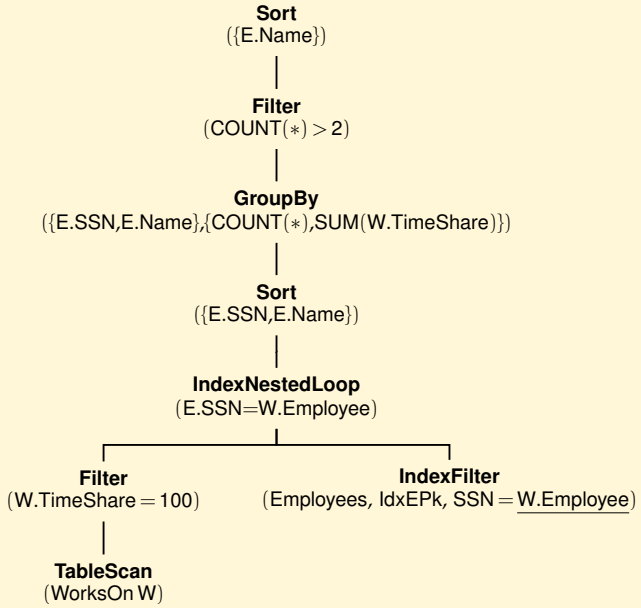
4) **SELECT** E.SSN, E.Name, COUNT(*), SUM(W.TimeShare)
FROM Employees E, WorksOn W
WHERE E.SSN = W.Employee **AND** W.TimeShare = 100
GROUP BY E.SSN, E.Name
HAVING COUNT(*) > 2
ORDER BY E.Name;



Logical query plan



Physical query plan without indexes



Physical query plan with indexes