CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

More Relational Algebra

LECTURE 6

Dr. Philipp Leitner

philipp.leitner@chalmers.se

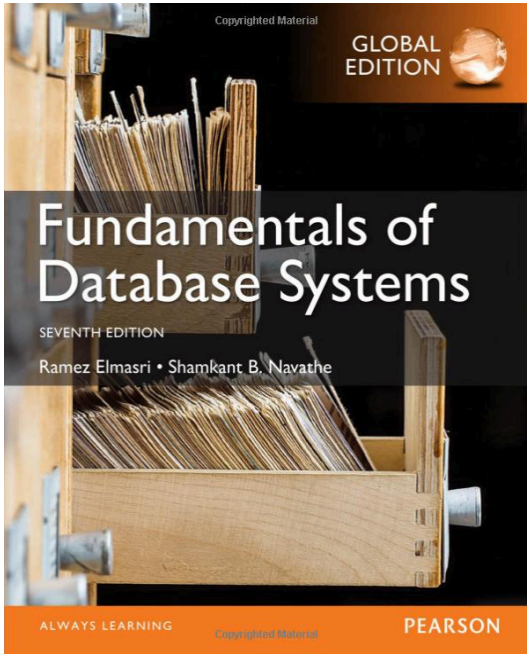
@xLeitix

LECTURE 6

Covers …

Parts of Chapter 8
Parts of Chapter 14 (high-level!)

Please read this up until next lecture!



What we will be covering

- Aggregation
- Outer Joins
- Basics of Functional Dependencies
- Normalization

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$ OR $R_1 * R_2$

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$

Let’s do some in-class exercises.

Figure 5.6

One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

“Find the first and last names of all employees, along with their department (number).”

Figure 5.6

One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

“Find an unique set of last names of all employees born after 1959”

Figure 5.6

One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

"Find the first and last names of all employees born after 1959, and store them to a relation NEWBIES(First,Last)"

Figure 5.6

One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

“Find all pairs of employees with different sex.”

(for simplicity it’s ok if the same pairs appear multiple in the result)

Figure 5.6

One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

“Find the last names of all supervisors.”

Aggregation

What the operations we discussed so far cannot express are **aggregations**

Functions that calculate something **across multiple tuples**

SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT

Aggregate Functional Operation $\mathscr{F}$

General notation:

$$\texttt{<grouping>}\mathscr{F}\texttt{<functions>}(R)$$

We'll talk in a second about what the "grouping" is about

whereas `<functions>` is a list of

`[MIN|MAX|AVERAGE|SUM|COUNT] <attribute>`

Usage Examples

$\mathcal{F}_{\texttt{MAX Salary}}(\texttt{EMPLOYEE})$

Retrieves the maximum salary of an employee

$\mathcal{F}_{\texttt{MIN Salary}}(\texttt{EMPLOYEE})$

Retrieves the minimum salary

$\mathcal{F}_{\texttt{SUM Salary}}(\texttt{EMPLOYEE})$

Retrieves the sum of all salaries

$\mathcal{F}_{\texttt{COUNT Ssn, AVERAGE Salary}}(\texttt{EMPLOYEE})$

computes the count (number) of employees and their average salary

Result of Aggregate Queries

Aggregate queries (without grouping) are different to what we discussed so far because they essentially return **single values**

Still in form of a relation for completeness, but as a relation with a **single tuple**

$\mathcal{F}_{\texttt{COUNT Ssn, AVERAGE Salary}}(\texttt{EMPLOYEE})$

Count_ssn	Average_salary
8	35125

Result of Aggregate Queries

We cannot directly combine aggregate queries and normal projection

$\mathcal{F}_{\texttt{COUNT Ssn, }\sout{\texttt{Salary}}}(\texttt{EMPLOYEE})$

Note that applying any operation, including projection or renaming, **to the result relation** is fine:

$\rho_{(\texttt{NR\_OF\_EMPLOYEES, MAX\_SAL})}(\mathcal{F}_{\texttt{COUNT SSN, MAX Salary}}(\texttt{EMPLOYEE}))$

The Special Case of COUNT

COUNT has one special property - it generally does not matter **which** attribute in a given relation you count over. The result is always the same (the number of tuples in the relation).

Note that COUNT does **nothing clever** - it does not remove duplicates, nor does it skip NULL values.

Handling Duplicates in Aggregate Functions

Recall that relations are **not allowed to have duplicate tuples**

All duplicates are removed from the results of relational operations to keep this constraint intact

However, **during processing** duplicates are fine

For instance, $\mathcal{F}_{\texttt{AVERAGE Salary}}(\texttt{EMPLOYEE})$ **does not** remove duplicates before calculating the average

In practice this matters only for SUM, COUNT, and AVERAGE

Grouping

Often we don't want to aggregate over the **entire** relation, but first group the tuples by some common attribute

Example:

Find the number and average employee salary **per department**

Basic idea:

For each unique department number, find all employees that work in that department and apply aggregate functions to those

Grouping

$$_{\text{DNO}}\mathcal{F}_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$$

This operation **groups** employees by DNO (department number) and computes the count of employees and average salary per department

Result is a relation with one tuple per DNO, and two additional attributes (the count and average salary for each DNO)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

OUTER JOINS

In NATURAL JOIN and EQUIJOIN, tuples without a matching (or related) tuple are **eliminated from the join result**

Including tuples with null in the join attributes

A set of operations, called OUTER JOINS, can be used when we want to keep all the tuples, regardless of whether or not they have matching tuples in the other relation.

Regular JOINS can lead to information loss

Example:

$$\texttt{DEPARTMENT} \bowtie_{\texttt{MGRSSN=SSN}} \texttt{EMPLOYEE}$$

If a department currently does not have a manager (for whatever reason) it won't show up in the result relation.

Basically always an issue when one or both of the join attributes are allowed to be NULL

Types of OUTER JOINS

LEFT OUTER JOIN (keeps every tuple in the **left** relation)

RIGHT OUTER JOIN (keeps every tuple in the **right** relation)

FULL OUTER JOIN (keeps every tuple in **both** relations)

Example OUTER JOIN

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{EMPLOYEE} \bowtie_{\text{EMPLOYEE.DNO=DEPARTMENT.DNUMBER}} \text{DEPARTMENT})$$

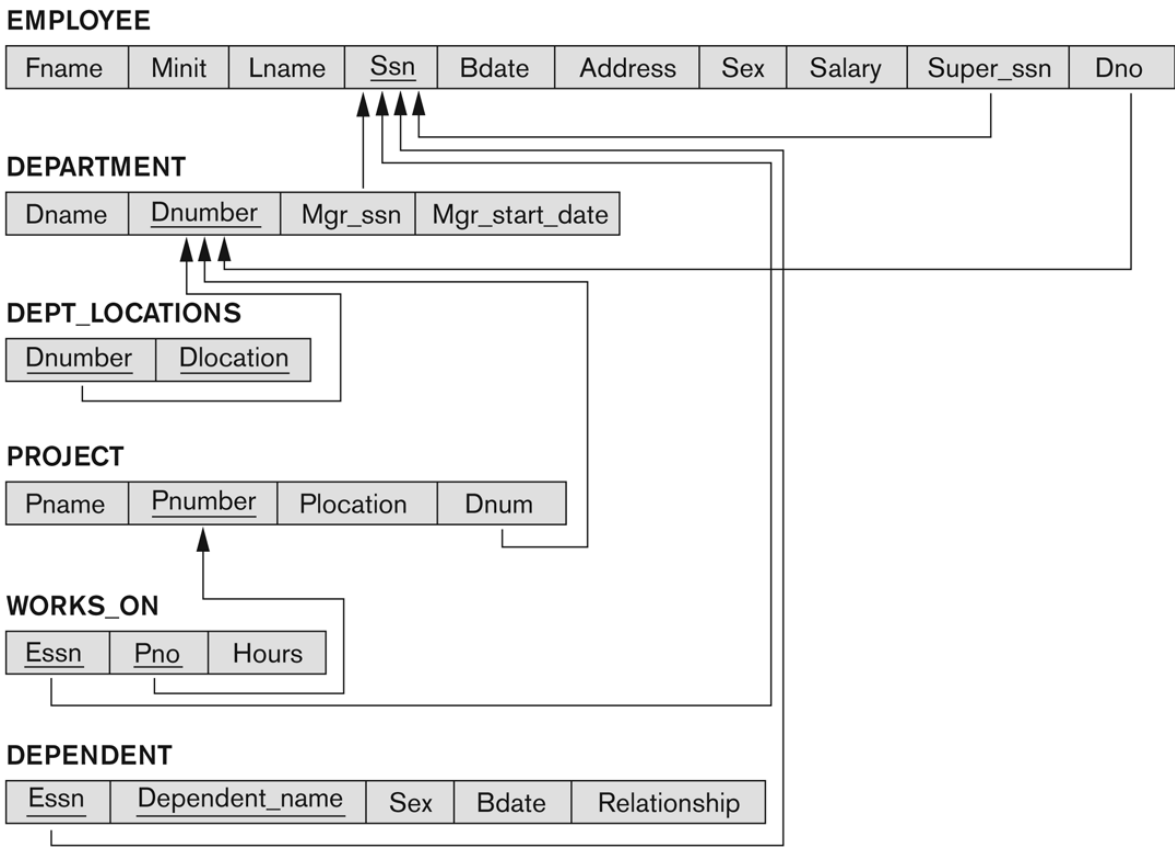
RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

Some more exercises.

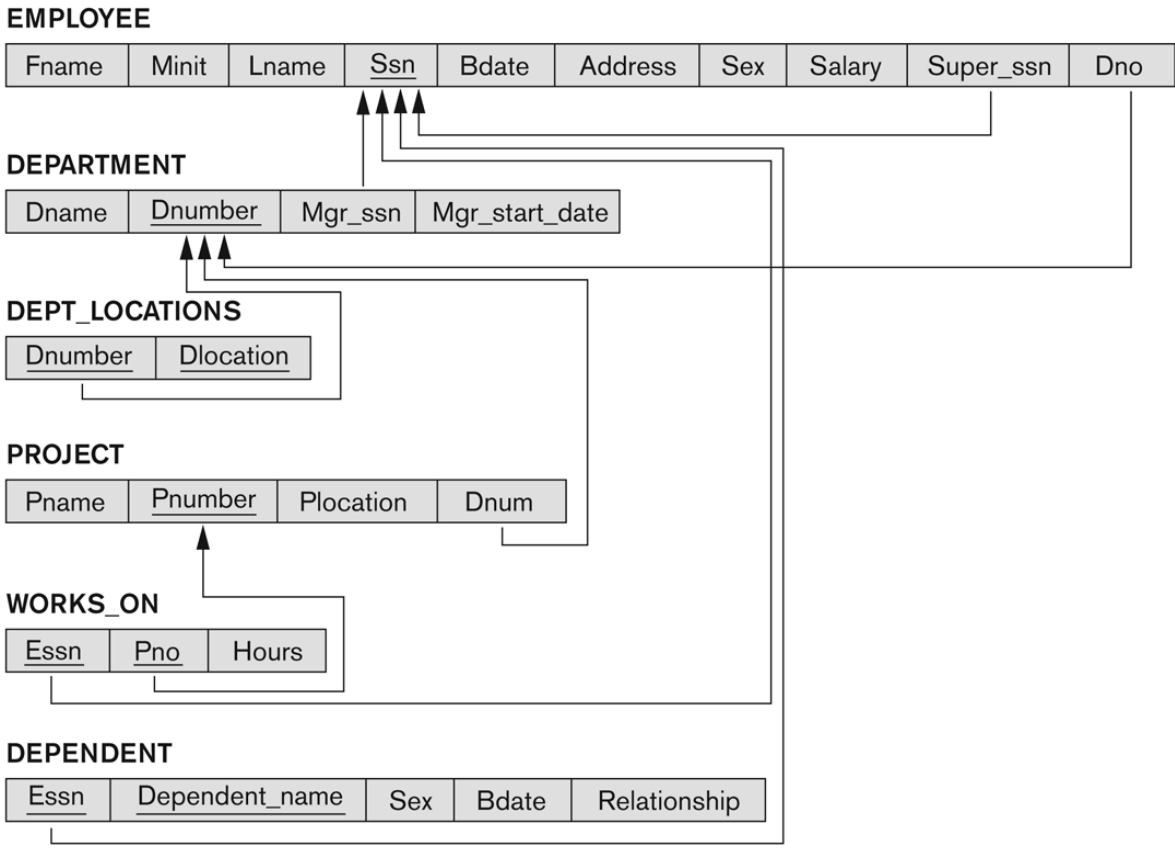
How many projects are there at each location?

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



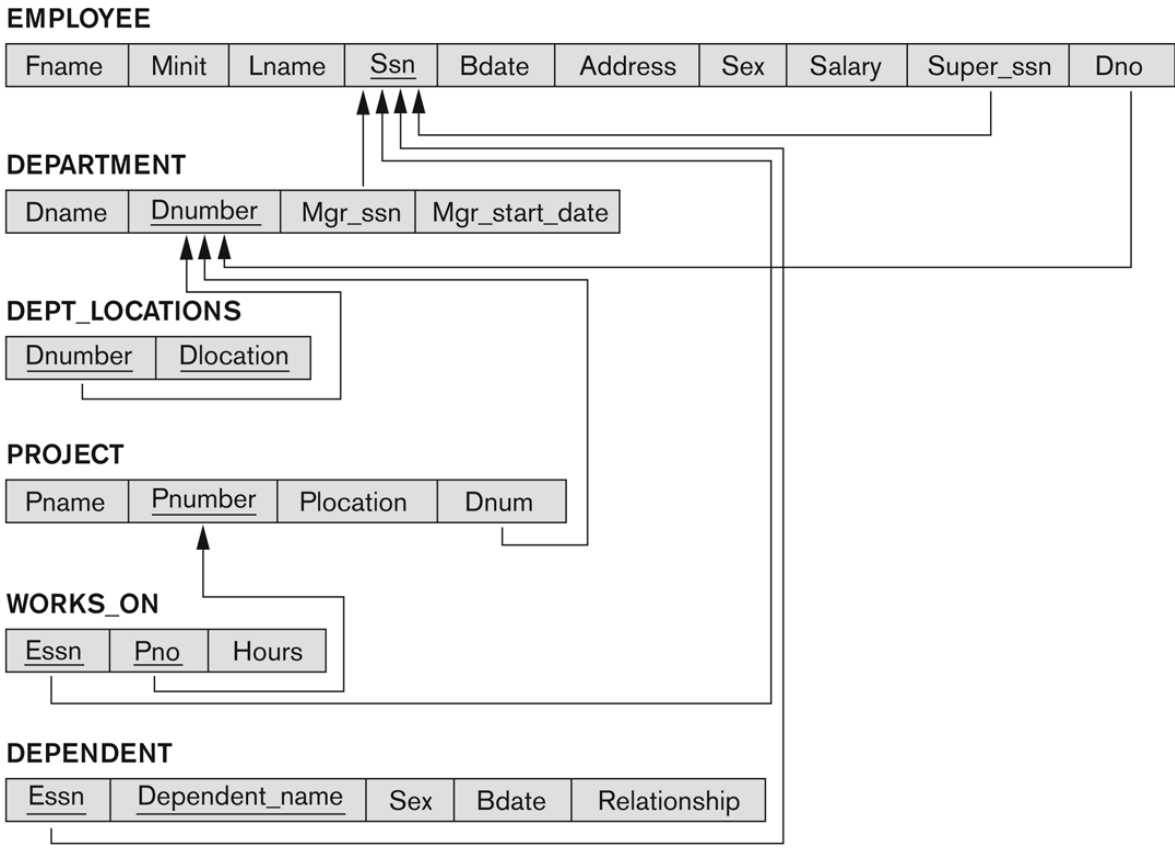
Get a list of all projects and the employees (all info, not just the employee Id) that work on them. Projects with no employees working on them should still be listed.

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



For each project name, find out how many employees are assigned to them. Projects without employees *should not* show up in the results (result should be a list of project names and employee counts).

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Let’s take a big step back now!

What constitutes a “good” relational design?

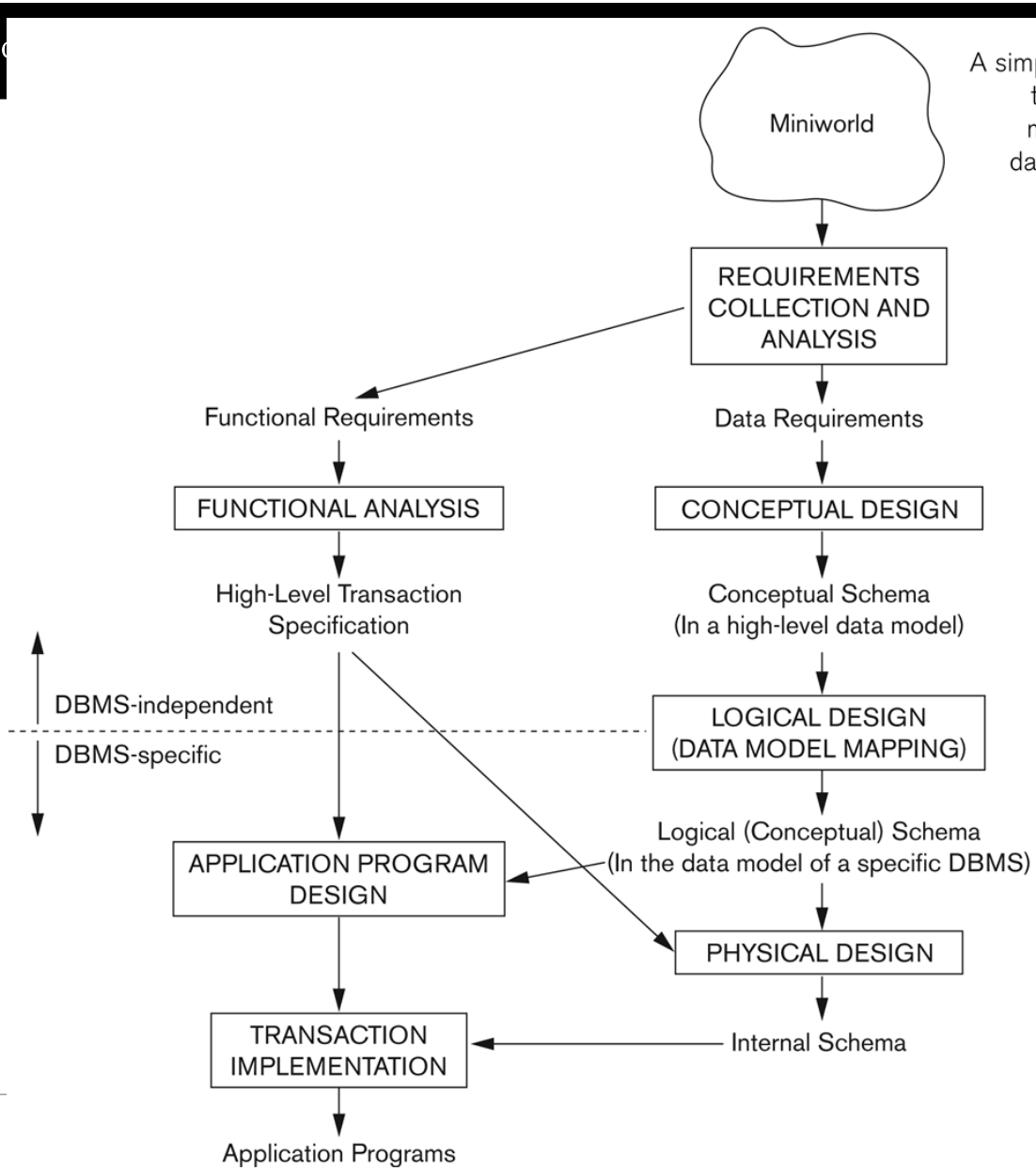


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Informal criteria for “good” design

Clear **semantics**

Mapping to the real world

No (or at least controlled) **redundancy**

Avoidance of NULL values

Support for **arbitrary queries**

Efficiency

An example of redundancy

Example:

```
EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
```

(the employee and project name are **redundant**, because we could also get them via join to the employee and project relations)

Informally:

A redundancy happens if information is stored explicitly that could also be derived from some other place in the database.

On redundancy

Two problems of redundancy:

(1) **wastes space** (same info is stored multiple times)
This is nowadays not the biggest problem anymore

(2) data can easily become **corrupted** when updating
(update anomalies)

On redundancy

However:

There is usually a trade-off, and in some database designs developers will choose **controlled redundancy** to improve query times.

Joins, and even moreso calculations, are expensive, but projections are cheap.

Alternatively:

Using **database views** (a “virtual relation” where certain information is pre-calculated for faster querying)

Supporting arbitrary queries

It is possible to design a database in a way that certain queries become **technically impossible** or **deliver wrong results** (spurious tuples)**.**

This is usually the case if the relations contain matching attributes that are not primary key / foreign key pairs.

This has a lot to do with the notion of **functional dependencies.**

Functional dependencies

Functional dependencies (FDs)

- Used to specify formal measures of the **"goodness"** of relational designs
- And keys are used to define **normal forms** for relations
- Are constraints that are derived from the **meaning and interrelationships** of the data attributes

A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

Functional dependencies

A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

Mathematically: X → Y (“X implies Y”)

Or in RA form: t1[X]=t2[X] → t1[Y]=t2[Y]

Some examples

Social security number determines employee name

$$SSN \rightarrow ENAME$$

Project number determines project name and location

$$PNUMBER \rightarrow \{PNAME, PLOCATION\}$$

Employee ssn and project number determines the hours per week that the employee works on the project

$$\{SSN, PNUMBER\} \rightarrow HOURS$$

FDs and keys

FDs are **not bidirectional**

- `SSN` → `ENAME`, but the inverse is not true

Keys **automatically** have a FD on every attribute in the relation

- If you know the key, you can determine all other attribute values

Redundancies formulated as FDs

Note that data redundancies can be formulated as (unwanted) functional dependencies:

```
EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
```

```
EMP_PROJ[Ename] → EMP[Ename]
EMP_PROJ[Pname] → PROJECT[Pname]
```

These are said to be **redundant FDs** as there are also the expected FDs between primary / foreign keys.

Colloquially, we can state that we need to maintain multiple pairs of FDs where one would do.

Finding FDs

We **cannot** define functional dependencies without knowing what the attributes in our data **mean**

However, given a valid database state, we can **rule out** certain FDs

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Normal Forms and Normalization

Normalization is the process of identifying FDs and refactoring a database schema so that (1) keys are properly identified and (2) unwanted FDs are removed.

Normal Forms:

A database is said to be in **normal form** after normalization.

Typically: 2NF, 3NF, BCNF

(there are more which we won't cover)

First Normal Form (1NF)

Disallows

- Composite attributes
- Multivalued attributes
- Nested relations

This basically comes automatically with the RM (RM does not support models that are not in 1NF)

Second Normal Form (2NF)

Requires that every attribute that is not part of a primary key is **fully functionally dependent** on the primary key.

That is, no attribute should have a FD on only a part of a composed primary key.

Third Normal Form (3NF)

Requires that no non-key attribute has a functional dependency on another non-key attribute.

Basically:

Avoid redundancies

More on normal forms

In the book you find (much) **more formal definitions** and concrete algorithms for normalization.

- Plus: more normal forms

In practice you will be fine if you follow the informal guidelines outlined earlier, most importantly:

- Define relations and attributes in a way that makes domain sense
- Stay aware of redundancy
- Define primary keys

Key Takeaways

Remaining concepts of RA:

- **OUTER JOINS**
- **Aggregate functions**

Normal forms and normalization:

- **Functional dependencies**
- **Normal forms (1/2/3NF)**