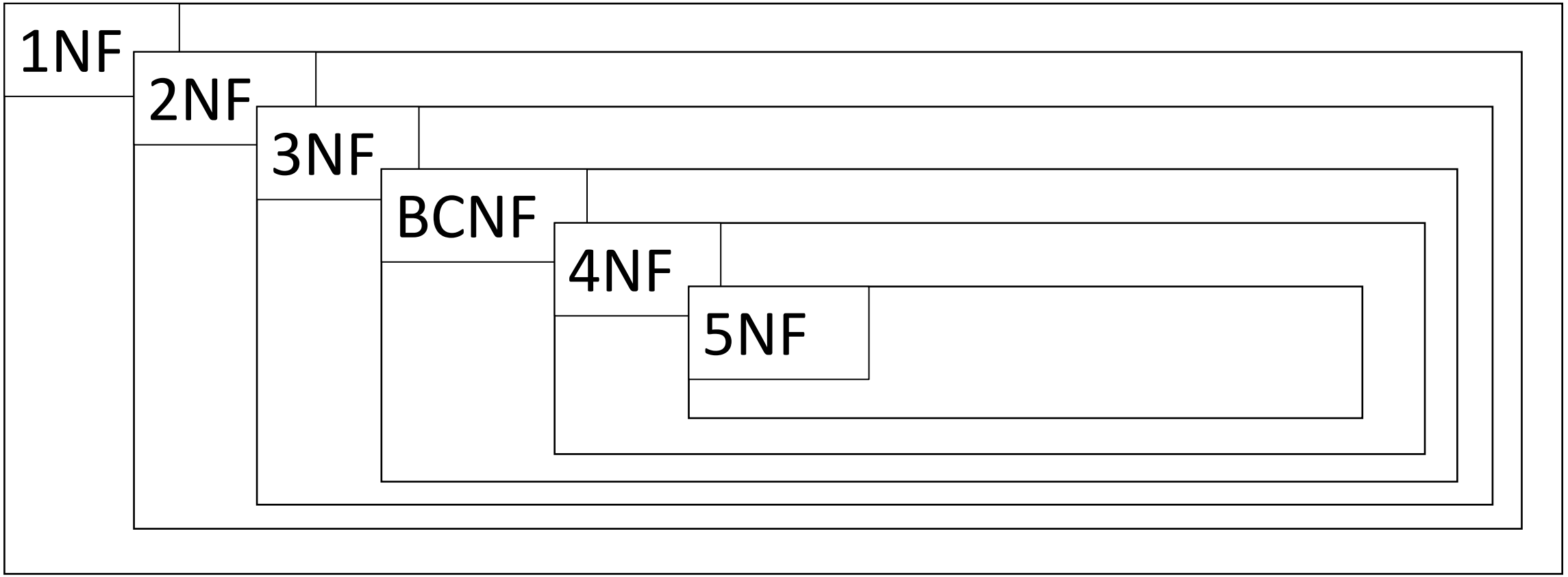
Functional Dependencies. Normal Forms

- there are multiple ways to model a data collection in the relational model (using tables)
- database design should be carried out in such a manner that subsequent queries and operations are performed as smoothly as possible, i.e.:
  - no additional tests are required when changing data
  - operations can be performed through SQL statements alone
- the above mentioned operations can be easily carried out when relations satisfy certain conditions (i.e., relations are in a certain normal form)

- the most common normal forms: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF
- they were defined by Codd (the first 3), Boyce and Codd (BCNF), and Fagin (4NF, 5NF)
- the following inclusions hold (for relations that satisfy various normal forms):



data redundancy

- generates a series of problems; some can be tackled by replacing a relation with a collection of smaller relations; each of the latter contains a (strict) subset of attributes from the original relation
- such decompositions can generate new problems (e.g., some data could be lost)
- ideally, only redundancy-free schemas should be allowed
- one should at least be able to identify schemas with redundancy, even if such schemas are allowed (e.g., for performance reasons)

- if a relation is not in normal form X, it can be decomposed into multiple relations that are in normal form X

Definition. The projection operator is used to decompose a relation. Let $R[A_1, A_2, \dots, A_n]$ be a relation and $\alpha = \{A_{i_1}, A_{i_2}, \dots, A_{i_p}\}$ a subset of attributes, $\alpha \subset \{A_1, A_2, \dots, A_n\}$. The projection of relation R on α is:

$$R'\left[A_{i_1}, A_{i_2}, \dots, A_{i_p}\right] = \Pi_\alpha(R) = \Pi_{\{A_{i_1}, A_{i_2}, \dots, A_{i_p}\}}(R) = \{r[\alpha] | r \in R\}$$

where $\forall r = (a_1, a_2, \dots, a_n) \in R \Rightarrow \Pi_\alpha(r) = r[\alpha] = \left(a_{i_1}, a_{i_2}, \dots, a_{i_p}\right) \in R'$,

and all elements in R' are distinct.

Definition. The natural join operator is used to compose relations. Let *R[α, β]*, *S[β, γ]* be two relations over the specified sets of attributes, $\alpha \cap \gamma = \emptyset$. The natural join of relations *R* and *S* is the relation:

$$R * S[\alpha, \beta, \gamma] = \left\{\left(\Pi_\alpha(r), \Pi_\beta(r), \Pi_\gamma(s)\right) \middle| r \in R, s \in S \text{ and } \Pi_\beta(r) = \Pi_\beta(s)\right\}$$

- a relation R can be decomposed into multiple new relations $R_1, R_2, \ldots, R_m$; the decomposition is good if $R = R_1 * R_2 * \cdots * R_m$, i.e., R's data can be obtained from the data stored in relations $R_1, R_2, \ldots, R_m$ (no data is added / lost through decomposition / composition)

- e.g., a decomposition that is not good:

Example 1. LearningContracts[Student, FacultyMember, Course],

and two new relations obtained by applying the projection operator to this relation: SF[Student, FacultyMember] and FC[FacultyMember, Course]

  - assume the original relation contains the following values:

LearningContracts	Student	FacultyMember	Course
r1	s1	f1	c1
r2	s2	f2	c2
r3	s1	f2	c3

- using the Π and * operators, the following values are obtained for the 2 projections and their natural join:

SF	Student	FacultyMember
r1	s1	f1
r2	s2	f2
r3	s1	f2

FC	FacultyMember	Course
r1	f1	c1
r2	f2	c2
r3	f2	c3

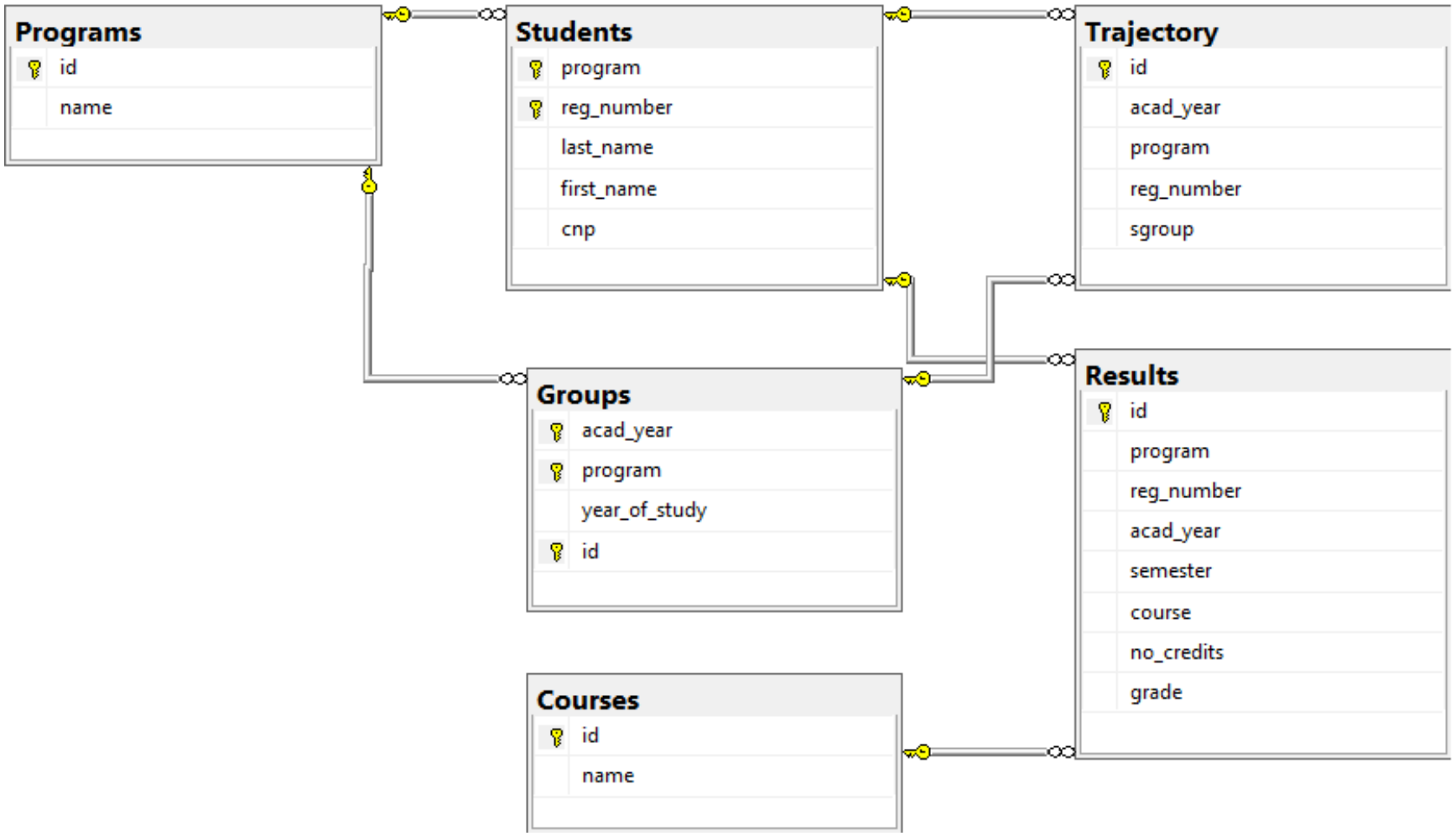
SF*FC	Student	FacultyMember	Course
r1	s1	f1	c1
r2	s2	f2	c2
?	s2	f2	c3
?	s1	f2	c2
r3	s1	f2	c3

SF*FC	Student	FacultyMember	Course
r1	s1	f1	c1
r2	s2	f2	c2
?	s2	f2	c3
?	s1	f2	c2
r3	s1	f2	c3

- the decomposition is not good, since the SF*FC relation contains extra records, i.e., records that were not present in the original relation

Appendix - More SQL Queries

- more query examples on the database:



- A student's trajectory (academic year and group).

```
SELECT acad_year, sgroup
FROM Trajectory
WHERE program = 2 AND reg_number = '7654'
```

- A student's grades.

```
SELECT acad_year, course, no_credits, grade
FROM Results
WHERE program = 2 AND reg_number = '7654'
```

- Students who belonged to group 915 in the academic year 2017-2018.

```
SELECT last_name, first_name, s.reg_number
FROM Students s INNER JOIN Trajectory t
  ON s.program=t.program AND s.reg_number=t.reg_number
WHERE acad_year=2017 AND sgroup='915'

SELECT last_name, first_name, s.reg_number
FROM Students s INNER JOIN
  (SELECT *
  FROM Trajectory
  WHERE acad_year=2017 AND sgroup='915') t
ON s.reg_number=t.reg_number AND s.program=t.program
```

- Students who belong to a group, but have no grades - in the academic year 2017-2018.

```
SELECT last_name, first_name
FROM Students AS s
WHERE EXISTS (SELECT *
          FROM Trajectory t
          WHERE acad_year=2017 AND t.program=s.program
               AND t.reg_number=s.reg_number)
     AND NOT EXISTS (SELECT *
                     FROM Results r
                     WHERE acad_year=2017 AND
                          s.program=r.program AND
                          s.reg_number=r.reg_number)
--ORDER BY last_name, first_name
```

- Students who belong to a group, but have no grades - in the academic year 2017-2018.

```
SELECT last_name, first_name
FROM (Students s INNER JOIN
        (SELECT *
        FROM Trajectory WHERE acad_year=2017) t
        ON s.program=t.program AND s.reg_number=t.reg_number
      )
      LEFT JOIN
        (SELECT *
        FROM Results
        WHERE acad_year=2017) r
          ON s.program=r.program AND s.reg_number=r.reg_number
WHERE grade IS NULL
```

- The number of students in the database.

```
SELECT COUNT(*) AS NoS
FROM Students
```

- The number of students born on the same day, regardless of year and month.

```
SELECT SUBSTRING(cnp,6,2) AS day, COUNT(*) AS NoS
FROM Students
GROUP BY SUBSTRING(cnp,6,2)
```

- The grades of a given student (only the maximum grade is required for each course).

```
SELECT course, no_credits, MAX(grade) AS maxgrade
FROM Results
WHERE program = 3 AND reg_number='virtualstudent11'
GROUP BY course, no_credits
--ORDER BY course
```

- The grades of a given student (only the maximum grade is required for each course). Include the name of the course in the answer set.

```
SELECT id, name, no_credits, maxgrade AS grade
FROM Courses c INNER JOIN
   (SELECT course, no_credits, MAX(grade) AS maxgrade
   FROM Results
   WHERE program = 3 AND reg_number='virtualstudent11'
   GROUP BY course, no_credits) r
ON c.id = r.course
--ORDER BY name
```

- For each student name that appears at least 3 times, retrieve all students with that name.

```
SELECT *
FROM Students
WHERE last_name IN
      (SELECT last_name
      FROM Students
      GROUP BY last_name
      HAVING COUNT(*)>=3)
--ORDER BY last_name, first_name
```

* rewrite the query without IN

- The number of students in each program and year of study in 2018.

```
SELECT g.program, g.year_of_study, COUNT(*) AS NoS
FROM Trajectory t INNER JOIN Groups g
 ON t.program=g.program AND t.sgroup=g.id AND
   t.acad_year=g.acad_year
WHERE t.acad_year=2018
GROUP BY g.program, g.year_of_study
```

- The last name, first name, program (id), registration number, number of passed exams, number of credits, and gpa for each student with at least 30 credits in passed courses at the end of 2017.

```
SELECT last_name, first_name, s.program, s.reg_number, COUNT(*)
  AS noc, SUM(no_credits) AS nocr,
  SUM(gradem*no_credits)/SUM(no_credits) AS gpa
FROM Students s INNER JOIN
  (SELECT program, reg_number, course, no_credits, MAX(grade)
    AS gradem
  FROM Results
  WHERE acad_year=2017 AND grade>=5
  GROUP BY program, reg_number, course, no_credits
  ) r
  ON s.program=r.program AND s.reg_number=r.reg_number
GROUP BY s.program, s.reg_number, last_name, first_name
HAVING SUM(no_credits)>=30
ORDER BY 3,1,2
```

- For every course, the number of grades and the number of passing grades in 2017.

* a. MySQL *

```
SELECT name, COUNT(*) AS nrg, SUM(IF(r.grade >= 5,1,0))
AS nrpg
FROM Courses c INNER JOIN
   (SELECT * FROM Results WHERE acad_year = 2017) r
   ON c.id = r.course
GROUP BY name
--ORDER BY 1
```

- For every course, the number of grades and the number of passing grades in 2017.

* b. Oracle *

```
SELECT name, COUNT(*) AS nrg, SUM(CASE WHEN (r.grade >= 5) THEN
1 ELSE 0 END) AS nrpg
FROM Courses c INNER JOIN
   (SELECT * FROM Results WHERE acad_year = 2017) r
   ON c.id = r.course
GROUP BY name
--ORDER BY 1
```

- * The grade average for each group in 2017. *

a. Maximum & passing grades in 2017.

```
CREATE VIEW studgrades AS
SELECT program, reg_number, course, no_credits, MAX(grade) AS
 mgrade
FROM Results
WHERE acad_year=2017 AND grade>=5
GROUP BY program, reg_number, course, no_credits
```

b. Students' grade average (in 2017).

```
CREATE VIEW studavg AS
SELECT program, reg_number,
  SUM(no_credits*mgrade)/SUM(no_credits) AS gradeavg
FROM studgrades
GROUP BY program, reg_number
HAVING SUM(no_credits) >= 30
```

- * The grade average for each group in 2017. *

c. Final query:

```
SELECT sgroup, AVG(gradeavg) AS gravg
FROM
   (SELECT program, reg_number, sgroup
   FROM Trajectory
   WHERE acad_year=2017) t
     INNER JOIN studavg AS s
       ON t.reg_number=s.reg_number AND t.program=s.program
GROUP BY sgroup
```

- what happens if two groups in different programs have the same id?

- * The grade average for each program in 2017. *

```
CREATE VIEW programsavg AS
SELECT id, name, AVG(gradeavg) AS pravg
FROM Programs p INNER JOIN studavg s ON p.id=s.program
GROUP BY id, name

SELECT * FROM programsavg
```