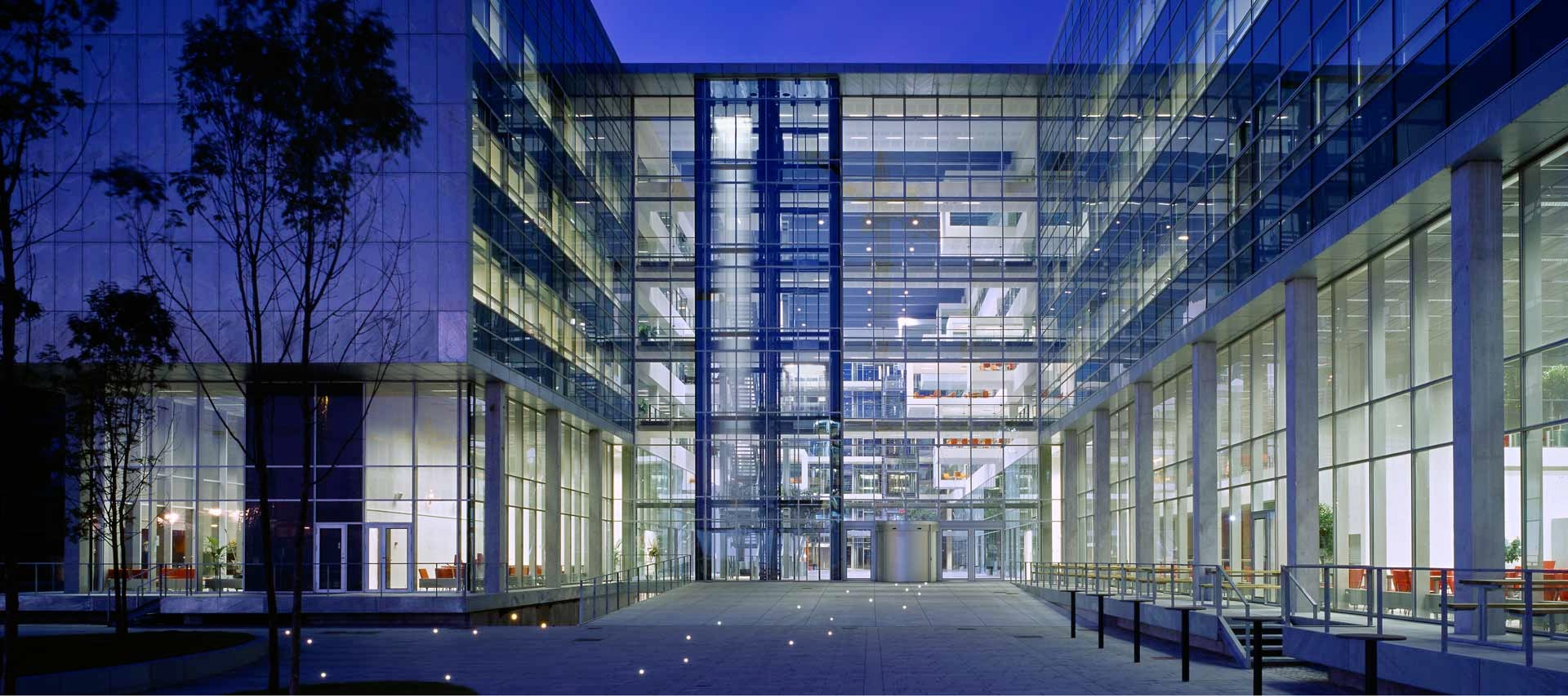

Week 6: Practical Normalization

Introduction to Database Systems

Slides from Björn Þór Jónsson

Redundancy Issues

- Consider the table Person(ID, Name, ZIP, City)

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø

- What can go wrong during:
  - Insert? – *what if new person lives in København V?*
  - Update? – *what if a municipality is renamed?*
  - Delete? – *what if Johan is deleted?*
  - Also: Extra storage

Solution: Decompose the Relation

Data Output | Explain | Messages | Notificatior

	id [PK] integer	name character varying	zip integer
1	1	Björn	2100
2	2	Johan	2300
3	3	Peter	2100

Data Output | Explain | Messages | N

	zip [PK] integer	city character varying
1	2100	København Ø
2	2300	København S

```
SELECT P.ID, P.Name, P.ZIP, Z.City
FROM Person P JOIN ZIP Z ON P.ZIP = Z.ZIP;
```

Data Output | Explain | Messages | Notifications

	id integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø

How to Decompose?

1. Add new relation
2. Fill new relation
3. Alter old relation

```
CREATE TABLE ZIP (
    ZIP INT PRIMARY KEY,
    City CHARACTER VARYING NOT NULL
);

INSERT INTO ZIP
SELECT DISTINCT ZIP, City
FROM Person;

ALTER TABLE Person ADD
      FOREIGN KEY (ZIP) REFERENCES ZIP(ZIP);
ALTER TABLE Person DROP COLUMN City;
```

How to Decompose – Homework Version

1. Add new relation(s)
2. Fill new relation(s)
3. Leave old relation unchanged!
   - Join of new relations should give same data as old relation!

```
-- CREATE and INSERT INTO ZIP as before
-- No ALTER TABLE statements

CREATE TABLE NewPerson (
    ID INT PRIMARY KEY,
    Name CHARACTER VARYING NOT NULL,
    ZIP INT NOT NULL REFERENCES ZIP(ZIP)
);

INSERT INTO NewPerson
SELECT ID, Name, ZIP
FROM Person;
```

Redundancy Issues Revisited

- Consider the table Person(ID, Name, ZIP, City)

Data Output | Explain | Messages | Notificatior

	id [PK] integer	name character varying	zip integer
1	1	Björn	2100
2	2	Johan	2300
3	3	Peter	2100

Data Output | Explain | Messages | N

	zip [PK] integer	city character varying
1	2100	København Ø
2	2300	København S

https://en.wikipedia.org/wiki/List_of_postal_codes_in_Denmark

- What can go wrong during:
  - Insert? – *what if new person lives in København V?*
  - Update? – *what if a municipality is renamed?*
  - Delete? – *what if Johan is deleted?*
  - Also: No extra storage

What Really Happened?

- Consider the table Person(ID, Name, ZIP, City)

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø

- Problem: **Functional Dependency** ZIP ⟶ City
  - If two records have the same ZIP value, they are **guaranteed** to have the same City value
  - ER does not capture this relationship easily
- Solution: **Decomposition**!

Towards Normal Forms

- This table is in **2NF**
  - Person: ID ⟶ Name
  - Person: ID ⟶ ZIP
  - Person: ID ⟶ City
  - **Person: ZIP ⟶ City** (transitive FD)

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø

- These tables are in **BCNF**
  - <key> ⟶ <attribute>
  - Person: ID ⟶ Name
  - Person: ID ⟶ ZIP
  - ZIP: ZIP ⟶ City

Data Output | Explain | Messages | Notificatior

	id [PK] integer	name character varying	zip integer
1	1	Björn	2100
2			
3			

Data Output | Explain | Messages | N

	zip [PK] integer	city character varying
1	2100	København Ø
2	2300	København S

- We like BCNF = no redundancy!

Discovering FDs: Inspection Method

- How can we find FDs?

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø

- Run a “thought experiment“:
  - Assume that an attribute is a key of a sub-relation
  - Consider which attributes could be in that relation?
- Study application design requirements:
  - Ex: Each vendor can only sell one part to each project
    - *vendor project* $\rightarrow$ *part*

Discovering FDs: SQL Method

- Study relation instances (when available)
  + Check application design requirements
  + Check reality ☺

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø

What is the SQL method?

- Assume that ZIP → City holds:
- For each ZIP value, how many different City values?

```
SELECT 'Person: ZIP --> City' AS FD,
CASE WHEN COUNT(*)=0 THEN 'MAY HOLD'
ELSE 'does not hold' END AS VALIDITY
FROM (
      SELECT P.ZIP
      FROM Person P
      GROUP BY P.ZIP
      HAVING COUNT(DISTINCT P.City) > 1
) X;
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø

- Let's check some more — code on LearnIT!

Discovering FDs: SQL Method

- Study relation instances (when available)
  - Can make a script to test all combinations, e.g.:
    - Use Java to write SQL queries into text file
    - Run the text file using psql
- Remember: Script can only say **MAY** HOLD!
  - Consider City → ZIP

+ Check application design requirements
+ Check reality ☺

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying	zip integer	city character varying
1	1	Björn	2100	København Ø
2	2	Johan	2300	København S
3	3	Peter	2100	København Ø