# COSC265 — Relational Database Systems

Neville Churcher

Department of Computer Science & Software Engineering
University of Canterbury

2021

# The Big Picture

## What we'll be covering this term

- Design — beyond (E)ER
- Looking under the hood

# Themes

Suggested readings are from Elmasri & Navathe 7th Edition

Real relational databases involve more than just data values and SQL. We will explore some of the factors which influence database quality & performance.

- ☆ Why bother with database design?
  - ★ Dependencies & normalisation (ch 14–15)
- ☆ Do we remember what "relational" means?
  - ★ Relational model/algebra/calculus refresher (ch 8)
- ☆ Does it matter how I ask?
  - ★ Query representation, decomposition & optimisation (ch 18–19)
- ☆ What happens when other people are using the database?
  - ★ Transactions & Concurrency control (ch 20–22)
- ☆ Is there life after `COMMIT`?
  - ★ Indexing, physical design (ch 16–17)

# Database Desiderata

- Abstraction — accurate model of 'real' world data and relationships
- CRUD operations transform DB from one *correct state* to another
- Integrity important
  - security, physical integrity
  - concurrency, recovery
  - design, semantic integrity
- Data independence

  Physical: block size, file location...

  Logical: independent user views

## General principles

- Avoid redundancy — store each fact only once
- Store each attribute in the right relation/table

# Informal guidelines

## Design guidelines

1. Each *tuple* in a relation should represent one *entity instance* or *relationship occurrence*.
2. Identify and remove potential *update anomalies*
3. Minimise likelihood of many *NULL* values
4. Natural join should not generate spurious tuples (*lossless join* condition)

# Update Anomalies

## What could possibly go wrong?

- Naïve solution to the supplier-parts problem with primary key $\{S\#, P\#\}$
- Much redundant data stored (Guideline 1 ignored)

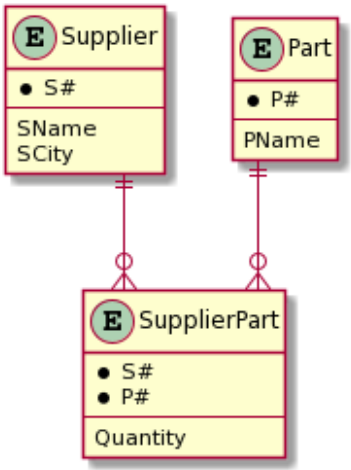
| S# | SName | SCity | P# | PName | QTY |
|---|---|---|---|---|---|
| 7 | Smith | Paris | 1 | spanner | 42 |
| 7 | Smith | Paris | 1 | spanner | 17 |
| 7 | Smith | Paris | 2 | hammer | 5 |
| 8 | Jones | Bonn | 1 | spanner | 20 |

**Insertion:** new supplier, say (9, Brown, York), cannot be added unless she supplies one or more parts — why not?

**Deletion:** deleting last order for supplier leads to loss of supplier details

**Update:** if Smith moves to Calais then redundant data leads to possibility of inconsistency

# A Better Solution?




| S# | SName | SCity |
|---|---|---|
| 7 | Smith | Paris |
| 8 | Jones | Bonn |

| P# | PName |
|---|---|
| 1 | spanner |
| 2 | hammer |

| S# | P# | QTY |
|---|---|---|
| 7 | 1 | 42 |
| 7 | 1 | 17 |
| 7 | 2 | 5 |
| 8 | 1 | 20 |

# Guideline 3: NULL values

- Physical storage considerations
- Possibly move attributes with high proportion of NULL values to a separate relation (with key)
- What does NULL really mean for the attribute?
  - Unknown?
  - Known to be empty, no value exists, ...
  - Inapplicable, invalid?

# Guideline 4: Lossy/Lossless joins

Project a relation into 2 narrower ones and join them back together . . .

- Start with $r_(R)$, where $R = ABC$
- Decompose into $r_1(R1)$ and $r_2(R2)$ where $R1 = AB$ and $R2 = BC$
- Re-join using $r' = r_1 \underset{B}{\bowtie} r_2$ — do we get our original relation back?
- We'll return to this soon . . .

| A | B | C |
|---|---|---|
| a1 | b1 | c1 |
| a3 | b1 | c2 |
| a2 | b2 | c3 |
| a4 | b2 | c4 |

$r_1$

| A | B |
|---|---|
| a1 | b1 |
| a3 | b1 |
| a2 | b2 |
| a4 | b2 |

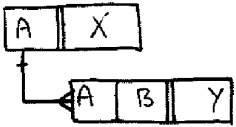
$r_2$

| B | C |
|---|---|
| b1 | c1 |
| b1 | c2 |
| b2 | c3 |
| b2 | c4 |

| A | B | C | |
|---|---|---|---|
| a1 | b1 | c1 | |
| a1 | b1 | c2 | ☆ |
| a3 | b1 | c1 | ☆ |
| a3 | b1 | c2 | |
| a2 | b2 | c3 | |
| a2 | b2 | c4 | ☆ |
| a4 | b2 | c3 | ☆ |
| a4 | b2 | c4 | |

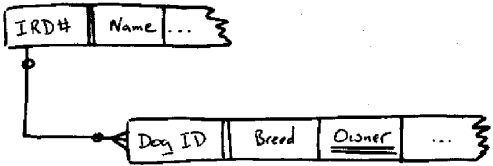
# Classifying Relationships

Ownership Relationship: Access path (FK) part of primary key.




The 'one' end is always mandatory — why?

Reference Relationship: The FK has the same domain as PK of other entity.

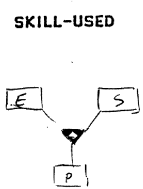



*Owner* has same domain as *IRD#*

The 'one' end may be optional — why is this?

# Ternary Relationships

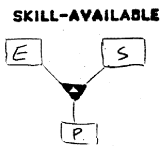
## *Genuine Ternary Relationship*




| EMP-NO | SKILL-NO | PROJ-NAME |
|---|---|---|
| 38 | 27 | GAMMA |
| 38 | 51 | GAMMA |
| 38 | 27 | DELTA |
| 38 | 3 | DELTA |

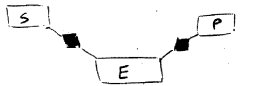
Genuine Ternary Relationship

## *Bogus Ternary Relationship*




| EMP-NO | SKILL-NO | PROJ-NAME |
|---|---|---|
| 14 | 22 | ALPHA |
| 14 | 22 | BETA |
| 14 | 35 | ALPHA |
| 14 | 35 | BETA |

reducible to...




EMP-SKILL

| EMP-NO | SKILL-NO |
|---|---|
| 14 | 22 |
| 14 | 35 |

EMP-PROJ

| EMP-NO | PROJ-NAME |
|---|---|
| 14 | ALPHA |
| 14 | BETA |

# Semantic Disintegrity and Connection Traps

## You can't get there from here

- Conceptual model, and any physical models derived from it, should accurately reflect the 'real' world
- Queries involve navigating paths from attribute to attribute
- Problems arise if necessary components (e.g. relationships) are omitted
- It should not be possible to misinterpret the data model (e.g. to form a meaningless query)
- Indirect relationships in data model must conform to real-world semantics
- Data modelling is independent of applications/queries but:
  - meaningful queries should be (correctly) answerable
  - meaningless queries should be detected and rejected

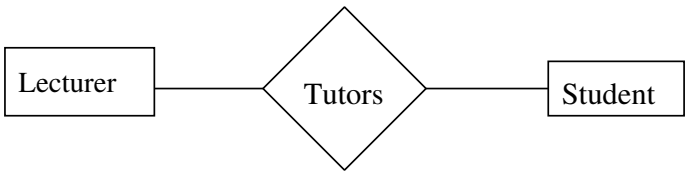
# Misinterpretation

## Names are important

Identifiers of components such as entities, attributes (particularly foreign keys) and relationships are important:

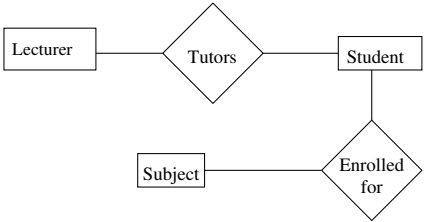
- ☆ for design of physical database
- ☆ for query formation — both procedures and *ad hoc*

Example (Subject or personal tutorship?)

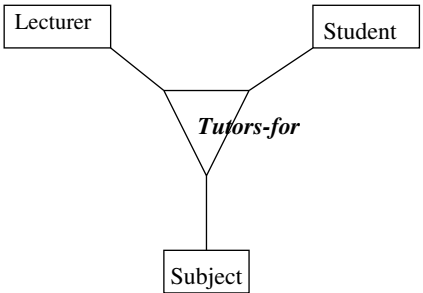


# Two possibilities

***Personal tutorship***



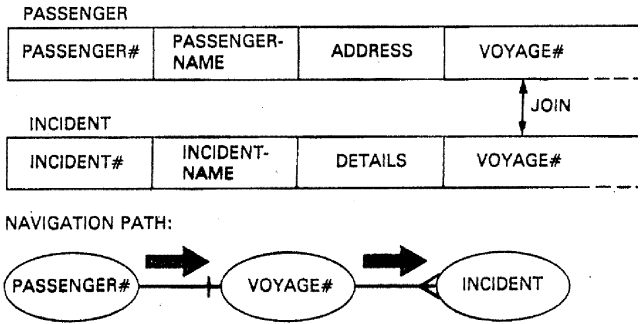

***Subject tutorship***



# Semantic Disintegrity

Apparently valid queries may involve joins or other operations which cause information to be lost.

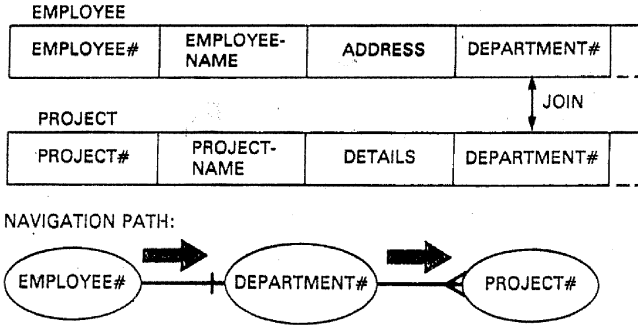
***A Valid Join*** Query: "List all the incidents that were reported on passenger Jones' voyage."




☆ Can this query produce the desired result?

# Semantic Disintegrity (continued)

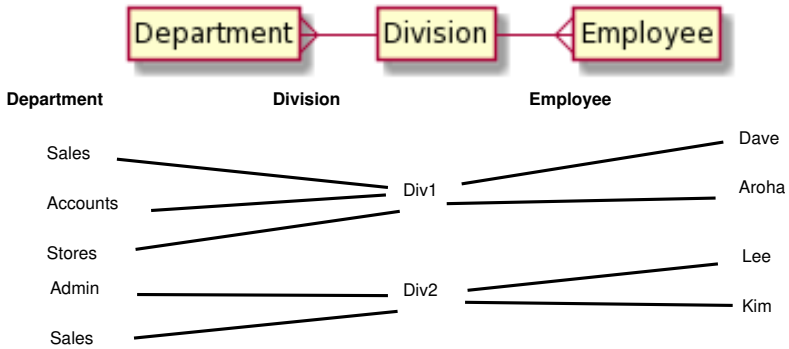
***An Invalid Join*** Query: "List all projects that employee Jones works on."




☆ Can this query produce the desired result?

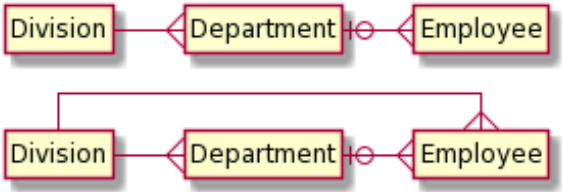
# Fan Traps

## Two 1:N relationships, occurrences fan out from 1 end




- Which employees belong to each department?
- What about employees attached to a division and not a department?

# Chasm Traps




- Original ambiguity (dept. of employee) resolved, but ...
- Need dummy department or extra relationship for employees with no department

## Homework Exercise

- Last term you produced a number of conceptual data models using EER diagrams and other table/relation identification techniques.
- Review the entities and corresponding relationships from your answers to some of the exercises in tutorials 1 & 2 — exercise 6 from tutorial 2 would be a good one to start with.
- Can you find any fan traps or chasm traps?
- Are there any queries your data model could not answer?

# Decomposition of Relation Schemes

## How do we produce designs?

- Update anomalies can be removed/reduced by decomposing relations (but remember Guideline 4!)
- Decomposition $\rho(R) = \{R_1, R_2, \ldots, R_k\}$ where $R = R_1 \cup R_2 \cup \ldots \cup R_k$
- If multiple decompositions are possible, which should we use?
- Top-down process — begin with single *universal relation*, $\mathcal{U}$, and perform *successive decompositions* (by projection) to produce "good" relations
- In practice, begin with set of relations resulting from analysis (e.g. EER) and decompose 'non-normal' relations
- Leads to more relations in database
- Alternative is *synthesis* — start with atomic facts and construct "good" relations

# But *how?*

- Model facts & constraints as data dependencies
- Use these dependencies (in decomposition or synthesis techniques) to produce relations with desirable properties
- Apply specific tests to assess schema quality
- Normal forms (Introduced by Codd in 1972) indicate quality level