Database Systems 2

Lecture 11

Database Design Methodology

Logical Design 1

Overview of the Methodology

Logical Database Design

2 Build and validate local logical data model for each user view

3 Build and validate global logical data model

Logical Database Design

2 Build and validate local logical data model for each user view

- 2.1 Map local conceptual data model to local logical data model
- 2.2 Derive relations from local logical data model
- 2.3 Validate model using normalisation
- 2.4 Validate model against user transactions
- 2.5 Draw Entity-Relationship diagram
- 2.6 Derive integrity constraints
- 2.7 Review local logical data model with user

2.1 Map local conceptual data model to local logical data model

- Remove M:N relationships
  - Introduce link tables
- Remove complex relationships (ternary or above)
  - Convert to binary relationships
- Remove recursive relationships
  - Possibly introduce an intermediate weak entity
- Remove relationships with attributes
  - Possibly introduce an intermediate weak entity
- Remove multi-valued attributes
  - Introduce a 1:M relationship with a separate entity
- Re-examine 1:1 relationships
- Remove redundant relationships
  - Can the same information be obtained through other relationships? If so, remove it. But be careful!

2.2 Derive relations from local logical data model

For each strong entity:

Create a relation (table) that includes all attributes and identify the PK.

For each weak entity:

Create a relation that includes all attributes, and include as a foreign key the PK of the owner entity. The PK of the weak entity is fully or partially derived from the owner entity. (Not just link tables!)

For each relationship:

Insert foreign keys in the appropriate tables as indicated by the cardinality and optionality of each relationship.

Document by using Skeleton tables.

2.3 Validate Model Using Normalisation

A procedure for deciding which attributes belong in which entity.

Organises the data according to its functional dependencies.

A normalised design is robust and free of update anomalies, with no redundant data.

Modern computers are getting increasingly powerful. It may be wise to implement a design that is easier to use at the cost of additional processing.

Normalisation produces a flexible design that can be easily extended.

Normalisation forces us to understand completely each attribute that is to be represented in the database. This benefit may be the most important.

The Rules of Normalisation

First Normal Form (1NF)

- Remove repeating groups to a separate relation, with a foreign key to preserve the link.

Second Normal Form (2NF)

- Remove partial dependencies on the primary key.

Third Normal Form (3NF)

- Remove transitive dependencies on the primary key.

Boyce-Codd Normal Form (BCNF)

- Removes remaining anomalies from functional dependencies.
- A 3NF table which does not have multiple overlapping candidate keys is also in BCNF

2.4 Validate model against user transactions

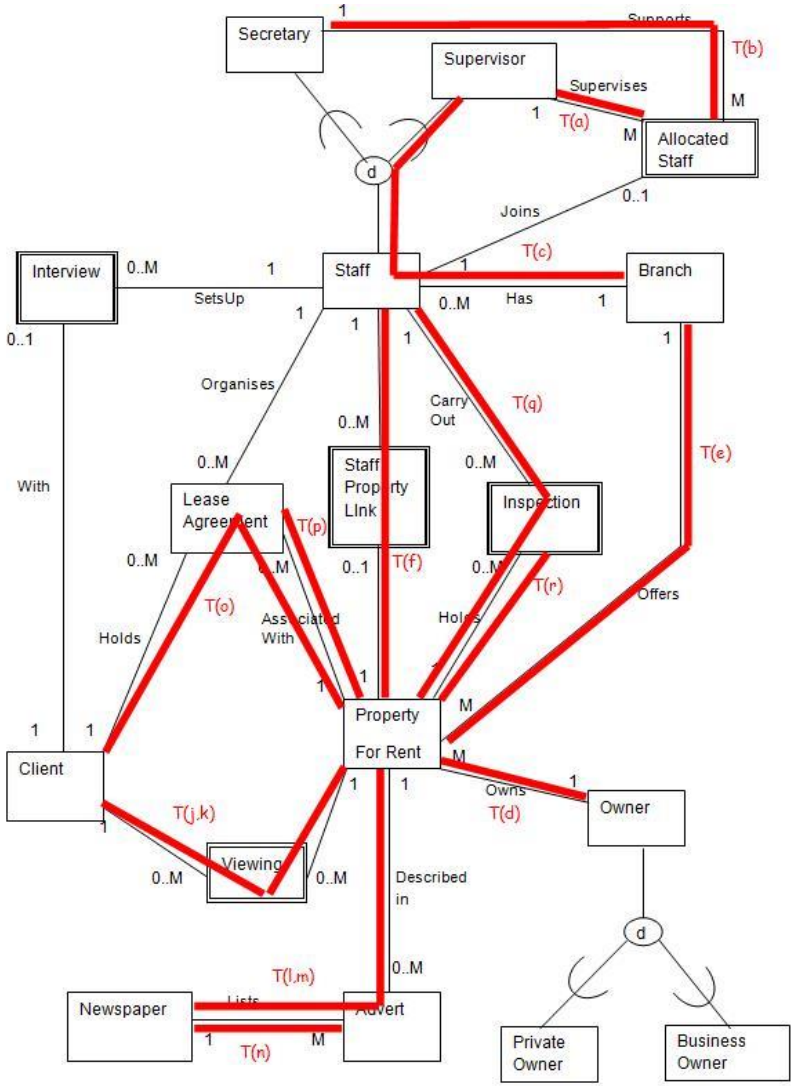
"Ensure that the local logical data model supports the transactions that are required by the user view"

You attempt to perform the transactions manually, using the ERD, attribute lists and primary/foreign key links.

This enables you to see if you have missed out any entities or relationships.

Two approaches:

- Check that all the information (entities, relationships, attributes) required by the transaction is provided by the model, by documenting a description of each transaction's requirements.
- Diagrammatically representing the pathway taken by the transaction on the ERD.


Secretary
Supervisor
Supervises
Allocated Staff
Supports
T(b)
T(a)
d
Joins
T(c)
Interview
SetsUp
Staff
Branch
Has
Organises
Carry Out
T(q)
T(e)
With
Staff Property Link
Lease Agreement
Inspection
T(p)
T(f)
T(o)
Associated With
T(r)
Offers
Holds
Property For Rent
Client
Owns
T(d)
T(j,k)
Owner
Viewing
Described in
T(l,m)
Newspaper
Lists
Advert
T(n)
Private Owner
Business Owner

2.5 Draw Entity-Relationship diagram

You should know how to do this by now.

Actually, this really ought to say a final ERD

.....so far

.....up to this point

.....for this particular user's view.

2.6 Derive integrity constraints

Constraints that we wish to impose in order to protect the database from becoming inconsistent.

Don't worry about whether your database package is able to do any of these, at the moment. This is a theoretical design.

Five types of integrity constraint:

- Required data
- Attribute domain constraints
- Entity integrity
- Referential integrity
- Enterprise constraints

Referential Integrity

If a foreign key contains a value, that value must refer to an existing primary key in the parent relation.

Several issues:

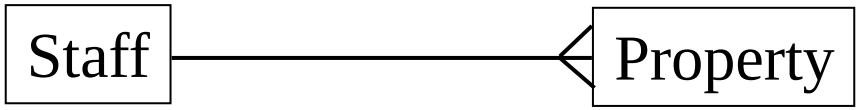
- Are nulls allowed in the foreign key fields?
  - This is one way of implementing optionality.

- How do we ensure referential integrity?
  - We must specify existence constraints.
  - These define conditions under which a primary key or foreign key may be inserted, updated or deleted.

Consider the following example:

Staff (staff_No, …)

Property (property_No, … , FK_staff_No *, ….)



Existence Constraints

Insert, Delete, Update on Child Relation

Case 1 - Insert row into child relation (Property)

- To ensure RI, check that the foreign key of the new property is either null or refers to an existing primary key in Staff.

Case 2 - Delete row from child relation (Property)

- RI is unaffected.

Case 3 - Update foreign key of child row (Property)

- Check that the new foreign key value is either set to null or refers to an existing primary key in Staff

Insert, Delete, Update on Parent Relation

Case 4 – Insert row into parent relation (Staff)

- No problems for RI. It becomes a parent without any children. Is this allowed by your logical design?

Case 5 – Delete row from parent relation (Staff)

- RI is lost if there exists a child row which referenced the deleted parent row. The designer must chose which of the following strategies is suitable:

No Action

- Prevent a deletion from the parent relation. Issue error message.

Cascade

- When a parent row is deleted, automatically delete any referenced child rows.
- What happens if the child row is acting as a parent for another relation?

Set Null

- When a parent row is deleted, the foreign key values in the child relation are all set to NULL.
- Are nulls in foreign keys allowed?

Set Default

- When a parent row is deleted, set foreign key values in all referenced child rows to their default values.
- Have default values been specified?

No Check

- Do nothing to ensure that RI is maintained.

Case 6 – Update primary key of parent row (Staff)

- As with deletion, RI is lost if there exist any child rows which referenced the old primary key value.
- One of the five strategies described above must be used to ensure RI is maintained.

For each foreign key, of each relation, you must define and document the existence constraints.

Existence Constraints Summary

Insert, Delete, Update on Child Relation

Case 1 - Insert row into child relation (Property)

Case 2 - Delete row from child relation (Property)

Case 3 - Update foreign key of child row (Property)

Insert, Delete, Update on Parent Relation

Case 4 – Insert row into parent relation (Staff)

Case 5 – Delete row from parent relation (Staff)

Case 6 – Update primary key of parent row (Staff)

Example of documentation

Property_for_Rent (Property_No, Street, Area, City, Postcode, Type,
Rooms, Rent, Owner_No*, Staff_No*, Branch_No*)

Primary Key Property_No

Foreign Key Owner_No **references** Private_Owner(Owner_No)
on delete NO ACTION **on update** CASCADE

Foreign Key Staff_No **references** Staff(Staff_No)
on delete SET NULL **on update** CASCADE

Foreign Key Branch_No **references** Branch(Branch_No)
on delete NO ACTION **on update** CASCADE.

2.7 Review Logical Data Model With User

Revise where necessary

Make sure they sign off on final logical model.

Should end up with many logical models, one for each group of users.