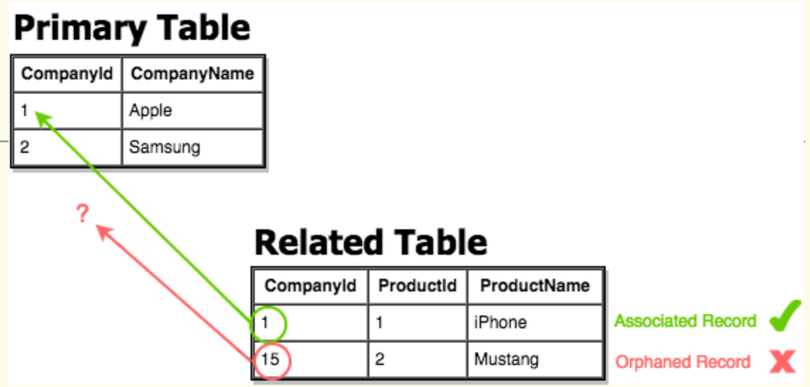
Logical Database Design 2



Logical Database Design for the Relational Model
Build and Validate Logical Data Model

- Recall that these are the steps involved in building and validating a Logical Data Model.
  - Step 2.1 Derive relations for logical data model.
  - Step 2.2 Validate relations using normalisation.
  - Step 2.3 Validate relations against user transactions.
  - Step 2.4 Define integrity constraints.
  - Step 2.5 Review logical data model with user.
  - Step 2.6 Merge logical data models into global model (optional step).
  - Step 2.7 Check for future growth.
- We have looked at step 1. We will look at the remainder now.

Topics List

- Validate relations using normalisation
- Validate relations against user transactions
- Define integrity constraints
- Review logical data model with user
- Merge logical data models into global model
- Check for future growth

Validate relations using normalisation

- Check composition of each table using the rules of normalisation, to avoid unnecessary duplication of data.

- Ensure each table is in at least 3NF.

Topics List

- Validate relations using normalisation
- Validate relations against user transactions
- Define integrity constraints
- Review logical data model with user
- Merge logical data models into global model
- Check for future growth

Validate relations against user transactions

- Ensure that the relations in the logical data model support the required transactions.
  - This type of check was carried out in Step 1.8 to ensure that the relations created in the previous step also support these transactions, and thereby ensure that no error has been introduced while creating relations.

- One approach is to examine transaction's data requirements to ensure that the data is present in one or more tables.

- If a transaction requires data in more than one table, check these tables are linked through the primary key/foreign key mechanism.

Topics List

- Validate relations using normalisation
- Validate relations against user transactions
- Define integrity constraints
- Review logical data model with user
- Merge logical data models into global model
- Check for future growth

Define integrity constraints

- Check integrity constraints are represented in the logical data model. This includes identifying:
  - Required data (e.g. Not NULL): Some attributes must always contain a value.
  - Attribute domain constraints: Every attribute has a set of values that are legal (domain).
  - Structural Constraints: Handled by relationship cardinality and participation.

Define integrity constraints

- Check integrity constraints are represented in the logical data model. This includes identifying (continued):
  - Entity integrity: Primary key must be unique and not null.
  - Referential integrity (see *Referential Integrity constraints).*
  - General constraints (Triggers): Updates to entities may be controlled by constraints governing transactions that are represented by the updates. For example, a property management system may have a rule that prevents a member of staff from managing more than 100 properties at the same time.
- Document all integrity constraints in the data dictionary.

Referential Integrity constraints

- As stated previously a foreign key value in one table is an attribute that is a primary key value in another table. Foreign key values are used to link tables together.

  Course(courseCode, title, leader, duration, points)
  Primary key courseCode

  Student(studentId, fName, lName, street, town, county, contactNumber, courseCode, year)
  Primary key studentId
  Foreign key courseCode references Course(courseCode)

Cascading Referential Integrity constraints

- We must ensure that cascading referential integrity constraints are specified that define conditions under which a candidate key or foreign key may be inserted, deleted, or updated:
  - **Insert tuple into child relation:** Value of foreign key must match primary key value of parent table or else be null. Inserting a *Student* record: *courseCode* must match an existing *courseCode* from the *Course* table or else be NULL.
  - **Delete tuple from child relation:** No difference or effect on parent table. Deleting a *Student* record: No effect on *Course* table.

Cascading Referential Integrity constraints

- Cascading referential integrity constraints (continued):
  - **Update foreign key of child tuple:** Value of foreign key must match primary key value of parent table or else be null. Updating a *Student* record: *courseCode* must match an existing *courseCode* from the *Course* table or else be NULL.
  - **Insert tuple into parent relation:** No difference or effect on child table. Inserting a *Course* record: No effect on *Student* table.

Cascading Referential Integrity constraints

- Cascading referential integrity constraints (continued):
  - **Delete tuple from parent relation:** This has repercussions on the child table. What if we deleted a Course that is associated with Students? The corresponding Student records would be enrolled on a Course (denoted by *courseCode*) that no longer exists in the *Course* table, therefore violates referential integrity.

Cascading Referential Integrity constraints

- **Delete tuple from parent relation** (continued):
  - There are a few options:
    - **No Action:** Prevent a deletion from parent relation if there are any referenced child tuples.
    - **Cascade:** When a parent tuple is deleted automatically delete any referenced child tuples.
    - **Set Null:** When a parent is deleted, the foreign key values in all corresponding child tuples are automatically set to null.
    - **Set default:** When a parent is deleted, the foreign key values in all corresponding child tuples are automatically set to their default values.
    - **No Check:** When a parent tuple is deleted, do nothing to ensure that referential integrity is maintained.

Cascading Referential Integrity constraints

- Cascading referential integrity constraints (continued):
  - **Update primary key of parent tuple:** Again, this has repercussions on the child table.
    What would happen if all course codes in the Course table were updated from 5 to 6 characters? For example, wd001 became seu001? Course codes in the Student table would no longer be valid.
    - If the primary key value of a parent relation tuple is updated, referential integrity is lost if there exists a child tuple referencing the old primary key value. To ensure referential integrity, the strategies as for (Delete tuple from parent relation) will suffice.

Cascading Referential Integrity constraints

Table	Operation	Can Violate Referential Integrity?	Why?	What can the DBMS do to prevent the violation?
Course (parent table)	Insert	No	A Course record can be created without reference to the Student table.	N/A
	Update	Yes	If the primary key of a Course record is changed, related records in the Student table will now reference a non-existing course.	1. Disallow the update. 2. Cascade the update. 3. Set the foreign key value to null. 4. Set the foreign key value to its default value (as long as the default value exists in the parent table - Course).
	Delete	Yes	If a Course record that is referenced in the Student table is deleted, the records in the Student table that previously referenced that Course, will now reference a non-existing Course.	1. Disallow the deletion. 2. Cascade the deletion. 3. Set the foreign key value to null. 4. Set the foreign key value to its default value (as long as the default. value exists in the parent table - Course).
Student (child table)	Insert	Yes	If a Student record is inserted, its courseCode value must reference an existing Course record.	Enforce Referential integrity
	Update	Yes	If the courseCode value in a Student record is updated, it must reference an existing Course record.	Enforce Referential integrity
	Delete	No	Deleting a Student record does not affect the Course table.	N/A

Cascading Referential Integrity constraints

- Note that the options available in MySQL are: *cascade*, *set null*, *no action*, and *restrict*. *No Action* is a keyword from standard SQL. In MySQL, it is equivalent to *restrict*.

Topics List

- Validate relations using normalisation
- Validate relations against user transactions
- Define integrity constraints
- Review logical data model with user
- Merge logical data models into global model
- Check for future growth

Review logical data model with user

- Review the logical data model with the users to ensure that they consider the model to be a true representation of the data requirements of the enterprise.

Topics List

- Validate relations using normalisation
- Validate relations against user transactions
- Define integrity constraints
- Review logical data model with user
- Merge logical data models into global model
- Check for future growth

Merge logical data models into global model

- Merge logical data models into a single global logical data model that represents all user views of a database.
- The activities in this step include:
  - Step 2.6.1 Merge local logical data models into global model.
  - Step 2.6.2 Validate global logical data model.
  - Step 2.6.3 Review global logical data model with users.

Topics List

- Validate relations using normalisation
- Validate relations against user transactions
- Define integrity constraints
- Review logical data model with user
- Merge logical data models into global model
- Check for future growth

Check for future growth

- Determine whether there are any significant changes likely in the foreseeable future and to assess whether the logical model can accommodate these changes.

- It is important to develop a model that is *extensible* and has the ability to evolve to support new requirements with minimal effect on existing users.