IF3111 – *Entity Integrity* dan *Referential Integrity*

Tricya Widagdo
Departemen Teknik Informatika
Institut Teknologi Bandung

Integrity Constraints

- The term *integrity* refers to the accuracy or correctness of data in the database.
- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- Integrity constraints is also known as *business rules*.

Domain Constraints

- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- New domains can be created from existing data types
  - E.g. **create domain** *Dollars* **numeric**(12, 2)
    create domain *Pounds* **numeric**(12,2)
- We cannot assign or compare a value of type Dollars to a value of type Pounds.
  - However, we can convert type as below
    (**cast** *r.A* **as** *Pounds*)
    (Should also multiply by the dollar-to-pound conversion-rate)

Domain Constraints (Cont.)

- The **check** clause in SQL-92 permits domains to be restricted:
  - Use **check** clause to ensure that an hourly-wage domain allows only values greater than a specified value.

    create domain *hourly-wage* **numeric(5,2)**
    constraint *value-test* **check**(*value* > = 4.00)
  - The domain has a constraint that ensures that the hourly-wage is greater than 4.00
  - The clause **constraint** *value-test* is optional; useful to indicate which constraint an update violated.
- Can have complex conditions in domain check
  - **create domain** *AccountType* **char**(10)
    constraint *account-type-test*
    check (**value in** ('Checking', 'Saving'))
  - **check** (*branch-name* **in** (**select** *branch-name* **from** *branch*))

Entity Integrity

- Entity Integrity Rule [DATE]:

  no component of the primary key of any base relvar is allowed to accept nulls.

- Ensures that each entity can be identified by using its primary key (whose values must not be missing).

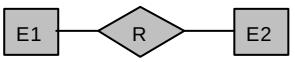
Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If "Perryridge" is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch "Perryridge".
- Formal Definition
  - Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2 respectively.
  - The subset α of R_2 is a ***foreign key*** referencing K_1 in relation r_1, if for every t_2 in r_2 there must be a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$.
  - Referential integrity constraint also called subset dependency since its can be written as

    $$\Pi_{\alpha}(r_2) \subseteq \Pi_{K1}(r_1)$$

Referential Integrity in the E-R Model

- Consider relationship set *R* between entity sets E_1 and E_2. The relational schema for *R* includes the primary keys K_1 of E_1 and K_2 of E_2.
  Then K_1 and K_2 form foreign keys on the relational schemas for E_1 and E_2 respectively.



- Weak entity sets are also a source of referential integrity constraints.
  - For the relation schema for a weak entity set must include the primary key attributes of the entity set on which it depends

Checking Referential Integrity on Database Modification

- The following tests must be made in order to preserve the following referential integrity constraint:

$$\Pi_{\alpha}(r_2) \subseteq \Pi_K(r_1)$$

- **Insert.** If a tuple t_2 is inserted into r_2, the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K] = t_2[\alpha]$. That is

$$t_2[\alpha] \in \Pi_K(r_1)$$

- **Delete.** If a tuple, t_1 is deleted from r_1, the system must compute the set of tuples in r_2 that reference t_1:

$$\sigma_{\alpha = t1[K]}(r_2)$$

  If this set is not empty
  - either the delete command is rejected as an error, or
  - the tuples that reference t_1 must themselves be deleted (cascading deletions are possible).

Database Modification (Cont.)

- **Update.** There are two cases:
  - If a tuple t_2 is updated in relation r_2 and the update modifies values for foreign key α, then a test similar to the insert case is made:
    - Let t_2' denote the new value of tuple t_2. The system must ensure that
      $$t_2'[\alpha] \in \Pi_K(r_1)$$
  - If a tuple t_1 is updated in r_1, and the update modifies values for the primary key (K), then a test similar to the delete case is made:
    1. The system must compute
       $$\sigma_{\alpha = t1[K]} (r_2)$$
       using the old value of t_1 (the value before the update is applied).
    2. If this set is not empty
       1. the update may be rejected as an error, or
       2. the update may be cascaded to the tuples in the set, or
       3. the tuples in the set may be deleted.