Integrity Constraints

- Accuracy or validity of data
  - No negative weights allowed for any item
- Ensure that the changes made by the users (updation, insertion, etc.) are *correct* and do not result in a loss of data consistency
- Prevent accidental damage to the database
- Key declarations as integrity constraints
  - Certain attributes form a key
  - Limit the set of legal insertions and updates to those that do not create two entities with the same value on a candidate key
  - Do not allow null values
  - Specified by the database administrator in some suitable language
  - Maintained in the system catalog or dictionary
- Forms of a relationship
  - A one-to-one or one-to-many relationship restricts the set of legal relationships among entities of a collection of entity sets
- Most of the current systems do not provide enough support to maintain integrity
- Integrity rules apply to base relations
  - Base relations reflect reality
  - Base relations must be constrained to contain correct configurations of values

Domain constraints

- Associate a set of values with every attribute
- Most elementary form of integrity constraints
- Tested whenever new data is entered or existing data is updated
- SQL allows us to define domains using a `create domain` clause as follows

```
create domain person_name char(20)
```

- Domain types
  - Possible for several attributes to have the same domain
    - Attributes `customer_name` and `employee_name` may have the same domain which is the set of all person names
    - Domain of `balance` and `branch_name` will certainly be different
    - Could the domain of `customer_name` and `branch_name` be the same?
    - View at conceptual v. physical level (`customer_name` v. `branch_name`)
  - Possible to test integrity of queries
- Domain types in SQL
  - SQL supports a restricted set of domain types

  - Fixed length characted string, with user specified length
  - Fixed point number, with user-specified precision
  - Integers (machine-dependent)
  - Small integer (machine-dependent)
  - Floating point (with user-defined precision)
  - Double precision floating point
  - Date type
  - Type coercion – casting from one type to another compatible type

- Null values
  - **not null** specification in domain declaration
  - Useful in specification of key attribute in a tuple

Referential Integrity

- A value that appears in one relation for a give set of attributes also appears for a certain set of attributes in another relation
- Basic concepts
  - Consider a pair of relations $r(R)$ and $s(S)$, and the natural join $r \bowtie s$
  - Let $\exists$ tuple $t_r \in r$ and $\nexists t_s \in s$ such that $t_r[R \cap S] = t_s[R \cap S]$
  - Dangling tuple – t_r
  - Suppose there is a student in the university database who is taking CS377 at present, but the course CS377 is not being offered
  - What if there is a class CS377 being offered but no student is enrolled in the class? Allowed (especially at the beginning of semester)
  - *Foreign key* – A [set of] attribute[s] that form a key in another relation but not in the current relation
  - Only tuples with a foreign key can be considered to be dangling tuples
  - *Referential integrity constraints*
    * Also known as *subset dependencies*
    * Let $r(R)$ and $s(S)$ be relations with primary keys R' and S'
    * A subset $\alpha \in S$ is a foreign key referencing R' in s if $\forall t_2 \in s, \exists t_1 \in r \mid t_1[R'] = t_2[\alpha]$
- Referential integrity in the E-R model
  - Every relation arising from the relationship set has integrity constraints
  - Weak entity sets have referential integrity constraints
- Database modification
  - May lead to violation of referential integrity
  - Consider the constraint

$$\Pi_\alpha(s) \subseteq \Pi_K(r)$$

    * This constraint can be described as α being a foreign key in the relation s referencing the set of attributes K in relation r
  - Test at *insert*
    * If tuple t_2 is inserted into s, the system must ensure that $\exists t_1 \in r \mid t_1[K] = t_2[\alpha]$. Or

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

- Test at *delete*
  - If t_1 is deleted from r, the system must select

$$\sigma_{\alpha=t_1[K]}(s)$$

    as the set of tuples in s that reference t_1
  - If this set is not empty, issue an error, or delete the set of selected tuples
  - May lead to cascading deletions
- Test at *update*
  - Updates to the referencing relation s
    - If t_2 is updated in relation s, and the update modifies the values for foreign key α, make a test similar to *insert*
    - Let t'_2 be the new value of t_2 after update

$$t'_2[\alpha] \in \Pi_K(r)$$

  - Updates to the referenced relation r
    - If t_1 is updated in r, and update modifies the values for primary key K, make a test similar to *delete*
    - Using the old value of t_1 (before updation), compute

$$\sigma_{\alpha=t_1[K]}(s)$$

    - If the set is not empty, reject the update as an error

- Referential integrity in SQL
  - Allows specification of primary and candidate keys and foreign keys as part of `create table` statement
  - Any attribute that is a member of the candidate key must be declared **not null**
  - Allowed to have null values in foreign key unless foreign key is a member of a candidate key

```
create  table deposit
        (branch-name     char(15),
        account-number   char(10)   not null,
        customer-name    char(20)   not null,
        balance          integer,
        primary key  (account-number, customer-name),
        foreign key  (branch-name)      references  branch,
        foreign key  (customer-name)    references  customer)
```

<u>Functional Dependencies</u>

- Generalization of the notion of *key*
- Basic concepts
  - Constraint on the set of legal relations
  - Given a relation R, attribute Y is *functionally dependent* on X, or $X \rightarrow Y$
  - X *functionally determines* Y
    - Each X-value in R has associated with it precisely one Y-value in R
    - In *supplier* relation, s# $\rightarrow$ (sname,status,city)
    - X is a candidate key in R
    - If X is primary key, then all attributes Y of R must be functionally dependent on X
  - Superkey

$$K \mid \forall(t_1, t_2) \in r,\ t_1 \neq t_2 \Rightarrow t_1[K] \neq t_2[K]$$

- Let $\alpha \subseteq R$ and $\beta \subseteq R$
- Functional dependency $\alpha \rightarrow \beta$ holds on R if in any legal relation $r(R)$,

$$\forall (t_1, t_2) \in r,\ \ t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- K is a superkey of R if $K \rightarrow R$
- In the scheme

$borrow\text{-}scheme =$ (branch-name, loan-number, customer-name, amount),
loan-number $\rightarrow$ amount
loan-number $\not\rightarrow$ customer-name

- Functional dependencies are useful in two ways:
  1. To specify constraints on a set of legal relations
  2. To test the legality of relations
- Consider the relations *parts*

Relation: Parts				
P#	P_Name	Color	Weight	City
p_1	Nut	Red	12	London
p_2	Bolt	Green	17	Paris
p_3	Screw	Blue	17	Rome
p_4	Screw	Red	14	London
p_5	Cam	Blue	12	Paris
p_6	Cog	Red	19	London

  Color $\not\rightarrow$ Weight
- Trivial functional dependencies
  * Satisfied by all relations
  * $A \rightarrow A$
  * A functional dependency of the form $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

• Closure of a set of functional dependencies

- Must consider *all* functional dependencies that hold
- Given a set of FDs F, can prove that other FDs hold
  * *logically implied* FDs
- Relation scheme $R = (A, B, C, G, H, I)$
- Set of functional dependencies

$$\begin{array}{lcl} A & \rightarrow & B \\ A & \rightarrow & C \\ CG & \rightarrow & H \\ CG & \rightarrow & I \\ B & \rightarrow & H \end{array}$$

- FD $A \rightarrow H$ is logically implied
- *Closure of* F is the set of all FDs logically implied by F
- Closure of F denoted by F^+
- Techniques for reasoning about FDs
  * Armstrong's axioms or Rules of inference
    · Reflexivity rule – If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
    · Augmentation rule – If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds

        · Transitivity rule – If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds

      The rules are *sound* because they do not generate any incorrect FDs.
      The rules are *complete* because for a given set F of FDs, they allow us to generate all of F^+

    * Additional rules
        · Union rule – If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
        · Decomposition rule – If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
        · Pseudotransitivity rule – If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds

- Closure of attribute sets
  - Compute the set of attributes functionally determined by the superkey α
  - Closure of attribute sets is the set of all attributes functionally determined by α under a set of FDs F
  - Closure of α is denoted by α^+
  - Algorithm to compute α^+

```
result := α;
while (changes to result) do
   for each FD β → γ in F do
      if β ⊆ result then
         result := result ∪ γ;
```

- Canonical cover
  - Used to minimize the number of FDs needed to be tested at update
  - Combine several FDs into one
  - Eliminate *extraneous attributes*
    * An attribute is extraneous in an FD if it can be removed without changing the closure of the set of FDs
    * Consider a set of FDs F and the FD $\alpha \rightarrow \beta$ in F
        · Attribute A is extraneous in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\} \cup \{(\alpha - A) \rightarrow \beta\}$
        · Attribute A is extraneous in β if $A \in \beta$ and the set of FDs $(F - \{\alpha \rightarrow \beta\} \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F
  - Canonical cover F_c for F is a set of dependencies such that F logically implies all dependencies in F_c and F_c logically implies all dependencies in F
  - Additional properties for F_c
    * No FD in F_c contains an extraneous attribute
    * Each left side of a FD in F_c is unique
  - Computation of canonical cover
    * Use union rule to replace the FDs of the form $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1\beta_2$
    * Test each FD $\alpha \rightarrow \beta$ for extraneous attributes in α
    * Test each FD $\alpha \rightarrow \beta$ for extraneous attributes in β
    * Repeat above steps till no more changes
  - Compute the canonical cover for the following F on scheme (A, B, C)

$$\begin{array}{lcl} A & \rightarrow & BC \\ B & \rightarrow & C \\ A & \rightarrow & B \\ AB & \rightarrow & C \end{array}$$

<u>Assertions</u>

- Predicate to express the condition to be always satisfied in the database

- Special constraints
  - Sum of all loan amounts for each branch must be less than the sum of all account balances at each branch
  - Every loan customer must maintain an account with a minimum balance of $1,000.00
- The **assert** statement
  - Simple assertion

```
assert  balance-constraint on deposit:
        balance ≥ 0
```

  - Assertion on two relations

```
assert  address-constraint on insertion to deposit:
    exists  (select  *
             from    customer
             where   customer.customer-name = deposit.customer-name)
```

<u>Triggers</u>

- Statement to be automatically executed as a side effect of DB modification
- Specifies
  - conditions under which trigger is to be executed
  - actions to be taken upon trigger execution
- Reading assignment – Section 5.5