Integrity Constraints

- Ensuring changes do not result in loss of consistency
  - Integrity constraints can be arbitrary predicates
    → very costly, limited in most DBMS

- Domain constraints
  - Domain types, similar to type checking in languages
  - OODB supports a rich set of domain types that can be extended easily
  - Certain attributes (usually keys) can be specified as not null to avoid anomalies

- Referential integrity
  - To ensure that a value that appears in one relation also appears in another relation
  → avoid dangling tuples
  - Notion of primary key and foreign key

Referential Integrity

Consider two relations \( r \) and \( s \), and their join \( r \times s \).
A dangling tuple: a tuple \( T_r \) in \( r \) that does not join with any tuple in \( s \).

<table>
<thead>
<tr>
<th>Branch</th>
<th>B-name</th>
<th>Assets</th>
<th>B-city</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_2 )</td>
<td>Perry</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deposit</th>
<th>B-name</th>
<th>Acc#</th>
<th>C-name</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>UVA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem if there’s no tuple in the Branch relation for UVA branch.
→ Integrity constraint must prohibit dangling tuples of this sort
Not all instances of dangling tuples cause problems.
→ No tuples in Deposit relation with Perry branch is OK
How to resolve it?
B-name is the primary key in the Branch relation
→ B-name in Deposit relation is a foreign key,
while B-name in Branch relation is not a foreign key.

Referential Integrity Constraints

Let \( r_1(R_1) \) and \( r_2(R_2) \) be relations with primary key \( K_1 \) and \( K_2 \), respectively.
A subset \( \alpha \) is a foreign key referencing \( K_1 \) in relation \( R_1 \), if it is requested that for every \( t_2 \) in \( r_2 \), there must be a tuple \( t_1 \) in \( r_1 \) such that \( t_1(K_1) = t_2(\alpha) \).

- Subset dependency: \( \Pi_\alpha(r_2) \subseteq \Pi_{K_1}(r_1) \)
  - Previous example, \( \Pi_{B-name}(Deposit) \subseteq \Pi_{B-name}(Branch) \)

- DB modification can cause violations of referential integrity
  → must be enforced for insert, delete, and update

  - To insert \( t_2 \) into \( r_2 \), there must be \( t_1 \) in \( r_1 \) such that \( t_1(K_1) = t_2(\alpha) \) or \( t_2(\alpha) \in \Pi_K(r_1) \)

  - To delete \( t_1 \) from \( r_1 \), compute the set of tuples in \( r_2 \) that reference \( t_1 \). If not empty, either reject the request or delete them all.

  - Update the referencing relation \( r_2 \): same as insert.
  - Update the referenced relation \( r_1 \): same as delete.

Assertions and Triggers

- Assertions
  - A condition DB must satisfy all the time
  - Testing whether any modification violates any assertion is very costly if complex assertions are allowed

SQL syntax: assert <name> on <relations>: <predicate>
SQL2 syntax: create assertion <name> check <condition>

- Trigger
  - A statement executed automatically by the system as a side effect of a modification to the DB
  → need to specify
  1) Condition under which the trigger is executed
  2) Actions to be taken

SQL syntax: define trigger <name> on <relations>: <condition>
  action_procedure <procedure name>
Assertions and Triggers: Examples

- **Assertions**

  ```
  create assertion salary_constraint
  check (not exists
    (select * from employee E, employee M, department D
     where E.salary > M.salary and
     E.dept_no=D.dept_no and D.manager_SSN=M.SSN))
  ```

- **Triggers**

  ```
  define trigger salary_trigger
  on employee E, employee M, department D
  E.salary > M.salary and E.dept_no=D.dept_no
  and D.manager_SSN=M.SSN
  action procedure inform_manager(D.manager_SSN)
  ```