9. Data Integrity

9.1 Introduction

Integrity constraints

- **Integrity constraints** (ICs)
  - are an instrument to ensure that changes of the database by authorized users do not lead to a loss of data consistency.
  - serve for a restriction of the database states to those ones that really exist in the real world.
  - are semantically derivable from the posed data model and can therefore already be specified during the creation of the schema.

- **advantages**
  - Consistency conditions are specified only once.
  - Consistency conditions are checked automatically by the DBMS.
  - APs do not need to care about a check of the conditions.

- **Static integrity constraints** relate to restrictions of the possible database states, **dynamic integrity constraints** to restrictions of the possible database state transitions.
– example for static IC: German professors may only have ranks C2, C3 or C4.
– example for dynamic IC: Professors may only be promoted, but not demoted. Their rank may not be set from C4 to C3, for example.

**Examples for ICs**

- No customer name may appear more than once in the relation “customers”.
- Each customer name in the relation “orders” must appear in the relation “customer”.
- No account of a customer is allowed to be less than USD -100.00.
- The account of Mr White may not be overchecked.
- Only those products can be ordered for which at least one supplier exists.
- The bread price may not be increased.
9.2 Static Integrity Constraints

**Type integrity / domain constraints**

- Each attribute value must be atomic.
- Determination of upper and lower bounds of attribute values
  - SQL92 allows the explicit definition of domains (not implemented in Oracle)
- Enumeration of values
  - Such enumeration types can be modeled by an own relation with one attribute.
- Null values
  - The construct **not null** attached to an attribute declaration can be used to achieve that the value of the corresponding attribute exists for all tuples.

**Tuple constraints**

- Restriction of the values which a tuple can take with respect to its different attributes
- Example: In a relation *occupancy*, where a tuple stores the reservation of a seat from a train station *X* to a station *Y*, *X* should be unequal to *Y*.
- In SQL a tuple constraint can be defined by the **check** clause during the specification of the schema.
Relation constraints / key constraints

- refers to the set of all tuples of a relation
- **key integrity** by means of primary key and candidate key
  - no two tuples with equal keys
  - in SQL declaration of this constraint with `unique` and `primary key` during the specification of the schema
  - No primary key or candidate key can have the value `null`.

- aggregation constraints
  - example: An aggregate value shall be above or below a predefined bound.

- Recursive constraints
  - example: In a relation `TrainConnection` each station should be connected with each other station.

Referential constraints

- refer to several relations
9.3 Referential Integrity

Features

- **Referential ICs** are conditions for relations which especially model a relationship.
- They are used to maintain the consistency between tuples of two relations.
- A referential IC requires that a tuple of a relation that refers to another relation has to refer to an *existing* tuple of that relation.
- Let $R$ and $S$ be relations with the schemas $R$ and $S$. Let $K \subseteq R$ be a primary key of $R$. Then $F \subseteq S$ is called **foreign key** of $S$, if for each data record $s$ of relation $S$ one of the following conditions holds:
  - $\forall A \in F : s[A] = \text{null}$.
  - There is a data record $r$ of $R$ such that $s[F] = r[K]$.
- *example:* relations `customers`, `products`, `orders`  
  *(m:n-relationship between `customers` and `products`)*
- possible problems if referential integrity is not fulfilled:
  - Customer orders products which do not exist
  - Products can be ordered by a customer who does not exist.
Maintenance of referential integrity

- In a relation, which models a relationship, we should ensure that values are assigned to the foreign keys.

- formulation of explicit integrity constraints in the query language, small extensions of the query language

- formulation in relational algebra:
  - relation \( R \) with primary key \( K \)
  - relation \( S \) with foreign key \( F \) that references \( K \)
    \[ \pi_F(S) \subseteq \pi_K(R) \]

- Allowed changes
  - insertion of a tuple \( s \) into \( S \) if \( s[F] \in \pi_K(R) \)
  - change of an attribute value of a tuple \( s \) if \( s'[F] \in \pi_K(R) \) (let \( s' \) be the new value of tuple \( s \))
  - change of \( r[K] \) of a tuple \( r \) if \( \sigma_{F=r[K]}(S) = \emptyset \)
  - deletion of a tuple \( r \) of \( R \) if \( \sigma_{F=r[K]}(S) = \emptyset \)
Referential integrity in the ER model

- $N$-ary relationships contain the keys of all participating entities, i.e., all keys are foreign keys.

- A weak entity set must include the primary key of the entity set on which the weak entity set depends. That is, the weak entity set contains the primary key of the strong entity set as foreign key.

Integrity constraints in SQL

- domain constraints
  - **not null**
    + null value as attribute value not permitted
    + example: FamilyName character(20) **not null**
  - **default** <value>
    + initialization of an attribute value with the value <value> which must be a constant or the result of the evaluation of an expression. The value <value> must be compatible with the domain of the attribute.
    + example: NoOfChildren smallint default 0
  - **default null**: optional, default value
Tuple constraint

- **check** clause as an integrity constraint that can be represented as a simple Boolean expression

- typical applications: range queries, realization of enumeration types, handling of null values

- example for range query: limitation that students may study for a maximum of 8 semesters.
  
  \[ \text{check sem between 1 and 8} \]

- example for enumeration type:
  
  \[ \text{check rank in ("C2", "C3", "C4")} \]

- handling of null values
  
  - Null value fits for **check** or foreign key constraints to arbitrary values. If a part of a foreign key is **null**, the key relates to all data which have the same defined part.
  
  - If a foreign key is to be totally **null** or totally defined, this must be explicitly expressed by a **check** clause.
example

\[ \text{check } ((S_1 \text{ is null and } S_2 \text{ is null and } \ldots) \text{ or } (S_1 \text{ is not null and } S_2 \text{ is not null and } \ldots)) \]

The constraint

\[ \text{check sem between 1 and 8} \]

is also fulfilled for a null value as semester.

Relation constraints/ key constraints

- primary key clause
  - automatically specified as not null
  - example:
    FirstName \textit{character}(20) not null,
    LastName \textit{character}(20) not null,
    Salary \textit{numeric}(11,2),
    \textbf{primary key} (LastName, FirstName)
candidate key: **unique**

- example:

  ```
  FirstName character(20) not null,
  LastName character(20) not null,
  **unique** (LastName, FirstName)
  ```

Referential Integrity

- foreign key: **foreign key**

- **unique foreign key** models a 1:1-relationship

- Foreign key must be marked as primary key in the referenced relation.

- example:

  ```
  create table customers(cid integer, ..., **primary key**(cid))
  create table orders(..., cid integer, ..., **foreign key**(cid) **references** customers(cid))
  ```

- standard behavior when violating a referential integrity constraint: rejection of the action that triggered the violation

- If a foreign key is specified with a **cascade** clause, changes of the respective primary key are propagated.
example

+ `create table` lectures ( ...
  ...,  
  `foreign key` held_by `integer references` professors(pers-id)
  `on delete cascade`
  `on update cascade`);

+ The deletion of a tuple with the key `pers-id` in `professors` has an effect on `lectures`. Also there the tuple is then deleted. This enables the realization of a dependent relationship.

+ A change of the attribute `pers-id` in `professors` is propagated to the foreign key `held_by` in `lectures`.

example

+ `create table` lectures ( ...
  ...,  
  `foreign key` held_by `integer references` professors(pers-id)
  `on delete set null`
  `on update set null`);

+ The attribute value `held_by` in `lectures` is set to `null`, if the referenced tuple with the key `pers-id` in `professors` is deleted.

+ The attribute value `held_by` in `lectures` is set to `null`, if the referenced tuple with the key `pers-id` in `professors` is changed.