# Part 7: Relational Model II

**References:**


- Elmasri/Navathe:Fundamentals of Database Systems, 3rd Edition, 1999.
  7.2 Relational Constraints and Relational Database Schemas
  7.3 Update Operations and Dealing with Constraint Violations
- Siberschatz, Korth, Sudarshan: Database System Concepts, 3rd Edition. McGraw-Hill, 1999.
  Section 6.2: "Referential Integrity".
- Kemper/Eickler: Datenbanksysteme (in German), Ch. 2, Oldenbourg, 1997.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Date/Darwen: A Guide to the SQL Standard, Fourth Edition, Addison-Wesley, 1997.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.






## Objectives

After completing this chapter, you should be able to:

- explain applications and problems of null values
  (i.e. reasons why to use them and why not).
- explain the concept of an outer join.
- explain the meaning of keys and foreign keys.
- explain CHECK constraints.
- check a given relational database schema with constraints for errors.
- develop relational database schemas for given applications.
  Normally, this is done via a translation form an ER-schema (see next chapter).





## Overview

1. Null Values
2. Constraints I: General, Keys
3. Constraints II: Foreign Keys/Referential Integrity





## Null Values (1)

- The relational model allows missing attribute values.
- So a tuple does not necessarily define values for all attributes of its schema. Table entries can be empty.
- Formally, all domains are extended by a new value "null".
  "Null" is not the number 0 or the empty string.
  It is different from all values of the data type.
- So if R has schema $(\mathsf{A}_1\colon \mathsf{D}_1, \ldots, \mathsf{A}_n\colon \mathsf{D}_n)$, then

$$\mathcal{S}(\mathsf{R}) \subseteq \big(|\mathsf{D}_1| \cup \{\mathsf{null}\}\big) \times \cdots \times \big(|\mathsf{D}_n| \cup \{\mathsf{null}\}\big).$$

- On the one hand, null values are often used in practice, on the other hand, they lead to quite severe problems.

## Null Values (2)

Null values are used in the following situations (among others):

- A value exists, but is not known.
  E.g., every customer's phone number may not be known, although probably most have a phone.
- No value exists.
  Not every customer has a fax.
- The attribute is not applicable to this tuple.
  For instance, storing the known programming languages only for the subclass of employees who are programmers.

A comittee once found 13 different meanings for a null value.





## Null Values (3)

Advantages of Null Values:

- In Oracle, null attribute values cost only one byte of storage space. Or no space at all if occurring at the row end.
- Without null values, it would be necessary to split most relations in many relations (corresponding to subclasses).
  E.g. `Customer_with_Phone`, `Customer_without_Phone`.
  Or extra relation: `Cust_Phone(CustNo, Phone)`.
  Null values can help to avoid joins and unions.
- If null values are not allowed, users will invent fake values to fill the missing columns.
  This makes the database structure even more unclear.





## Null Values (4)

Problems:

- Since the same null value is used for very different purposes, there can be no clear semantics.
- SQL uses a three-valued logic (true, false, unknown) for evaluating conditions in the presence of null values.
  For those accustomed to two-valued logic (most of us), there can be surprises — common equivalences do not hold.
- Most programming languages do not have null values.
  So when an attribute value is read into a program variable, it must be checked for a null value (→ indicator variables).





## "Not Null" Constraints (1)

- Since null values lead to complications, it can be specified for each attribute whether or not a null value is allowed.
  The relational model allows null values, but constraints that null values are invalid in certain columns can be specified.
- It is important to invest careful thought as to where null values are needed.
- Declaring many attributes "not null" will result in simpler programs and fewer surprises with queries.
- However, you flexibility is lost: Users are forced to enter values for all "not null" attributes.

# “Not Null” Constraints (2)

- In SQL, null values are allowed by default, and one must explicitly request “`NOT NULL`” in column definitions.
- However, often only a few columns can contain null values.
- In the simplified notation used here, attributes which can take a null value must be explicitly marked with a small “o” (optional) in the exponent:

  Instructor(Name, Room, OfficePhone, HomePhone$^o$)
- “Not null” constraints are assumed for all other attributes.





# “Not Null” Constraints (3)

- For tabular notation, the possibility of null values can be indicated in one of these ways:

| Instructor | | |
|---|---|---|
| Column | Type | Null |
| Name | char(30) | n |
| ⋮ | ⋮ | ⋮ |
| HomePhone | char(13) | y |

| Instructor | Name | Room | OfficePhone | HomePhone |
|---|---|---|---|---|
| Type | char(30) | char(9) | char(5) | char(13) |
| Null | n | n | n | y |





# Outer Join (1)

- The usual join eliminates tuples without a join partner:

| A | B |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |

$\bowtie$

| B | C |
|---|---|
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

$=$

| A | B | C |
|---|---|---|
| $a_2$ | $b_2$ | $c_2$ |

- The left outer join guarantees that tuples from the left table will appear in the result (filled with “null” if necessary):

| A | B |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |

⟕

| B | C |
|---|---|
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

$=$

| A | B | C |
|---|---|---|
| $a_1$ | $b_1$ | |
| $a_2$ | $b_2$ | $c_2$ |





# Outer Join (2)

- The right outer join preserves tuples from the right table:

| A | B |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |

⟖

| B | C |
|---|---|
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

$=$

| A | B | C |
|---|---|---|
| $a_2$ | $b_2$ | $c_2$ |
| | $b_3$ | $c_3$ |

- The outer join does not eliminate any tuples:

| A | B |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |

⟗

| B | C |
|---|---|
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

$=$

| A | B | C |
|---|---|---|
| $a_1$ | $b_1$ | |
| $a_2$ | $b_2$ | $c_2$ |
| | $b_3$ | $c_3$ |

# Overview

1. Null Values
2. Constraints I: General, Keys
3. Constraints II: Foreign Keys/Referential Integrity





# Constraints (1)

- Every database state should be a valid model of a possible situation in the real world. Often the declaration of tables and columns is not sufficient, and constraints must be added, i.e. conditions which every DB state must satisfy.
- The relational model contains special support for three common types of constraints:
  - keys,
  - foreign keys,
  - “not null” constraints.

  Some textbooks also list the domain constraint: Attribute values must be from the data type declared for that attribute.





# Constraints (2)

CHECK-Constraints:

- Another important class of constraints are CHECK-constraints.
- They, too, are contained in the SQL-89 standard and are supported in current systems (e.g. Oracle, DB2, SQL Server).
- CHECK-constraints specify conditions, which every tuple of a table must satisfy, e.g.:
  - `CHECK(YEAR_BORN >= 1000 AND YEAR_BORN < 3000)`
  - `CHECK(YEAR_BORN <= YEAR_DIED)`.
  - `CHECK(IS_CAPITAL = 'Y' OR IS_CAPITAL = 'N')`
- So CHECK constraints are specified by defining a selection condition $\varphi$ and requiring that $\sigma_\varphi(\mathrm{R}) = \mathrm{R}$ (all tuples satisfy $\varphi$).





# Constraints (3)

- The main restriction is that this condition can refer only to attributes of a single tuple.
- Since the condition can be evaluated locally for every single tuple in isolation, it can easily be checked whenever a tuple is inserted or modified.
- Normally, CHECK constraints count as satisfied if they access an attribute which is `NULL`.

  More precisely, they are only violated if they evaluate to the truth value “False”. If they evaluate to “Unknown”, the system will not reject the insertion/modification. The three-valued logic is explained below (Chapter about SQL).

# Constraints (4)

General Constraints:

- SQL92 has a "CREATE ASSERTION"-clause, which allows specifying general constraints.

  However, it is not supported in most/all(?) systems.

- General constraints should be specified formally enough so that application programmers know which conditions they have to check when data is entered.

  If constraints are specified as a query E which returns violations of the constraint (i.e. the constraint is $E = \emptyset$), this query can be run from time to time. See below (in the Part "Protecting Your Data") for more information.





# Keys (1)

- See above for a discussion of keys in the ER-Model.

  There is no essential difference between keys in the two models.

- A key of a relation R is an attribute A which uniquely identifies the tuples in R.

  I.e. there may never be two different tuples t and u in $\mathcal{S}(R)$ such that $t.A = u.A$.

- A key can also consist of several attributes ("composite key").

  If A and B together form a key, it is forbidden that there are two rows t and u which agree in both attributes (i.e. $t.A = u.A$ and $t.B = u.B$). They may agree in one attribute as long as they differ in the other.





# Keys (2)

Minimality of Keys:

- The usual definition of a key requires that the set of key attributes $\{A_1, \ldots, A_k\}$ is minimal.

  Since the unique identification property automatically holds for supersets, "nonminimal keys" are not interesting.

- This means that we cannot leave out any of the key attributes without destroying the property of unique identification.

- However, in this definition, a key is not only
  - a constraint, which excludes invalid database states,
  - but also an assertion about the real world, that certain states are possible.





# Keys (3)

Multiple Keys:

- A relation may have more than one key.

  For instance, {SSN} might be a key for an Employee-table, but {FirstName, LastName} might also be a key. Both keys are minimal, because neither is a subset of the other.

- One of the keys is designated as the "primary key".

  Other keys are called "alternate keys" or "secondary keys". SQL uses the keyword UNIQUE for alternate keys.

- The primary key attributes are underlined in the schema:

$$R(\underline{A_1 \colon D_1}, \ldots, \underline{A_k \colon D_k}, A_{k+1} \colon D_{k+1}, \ldots, A_n \colon D_n).$$

- Usually, the attributes of a relation are ordered in such a way that the primary key consists of the first attributes.

# Keys (4)

Keys and Null Values:

- The primary key cannot be null, other keys should not be null.

  It is as not acceptable if already the "object identity" of the tuple is not known.

- In SQL-89 and DB2, `NOT NULL` must be specified for every attribute in a `PRIMARY KEY` or `UNIQUE` constraint.
- In SQL-92 and Oracle, the "`PRIMARY KEY`" declaration automatically implies "`NOT NULL`", but "`UNIQUE`" does not.

  In Oracle, there can be several rows, all with a null value in a `UNIQUE` key. In SQL Server, only one row can be null.

  SQL-92 defines three different semantics for composite keys which have null values in only some of their attributes.





# Keys (5)

Keys and Updates:

- It is considered poor style if key attribute values are modified (updated).

  This would change the "object identity". Better: Delete the tuple first and then insert a tuple with the new values.

- But SQL does not enforce this constraint.

  The standard even contains specifications for what to do with foreign keys if the referenced key value is updated.

- Declaring a set of attributes as a key is a bit more restrictive than the unique identification property.

  However, the uniqueness is the main requirement for a key. Everything else is secondary.





# Keys (6)

The Weakest Possible Key:

- A key consisting of all attributes of a relation requires only that there can exist no two different tuples which agree in all attributes.
- Theoretically, relations are sets: Then this is no restriction.
- However, in practice, relations are multisets (bags), and this key ensures that there are no duplicates.
- Style Recommendation: Define at least one key for every relation in order to exclude duplicate tuples.
- If there is no other key, define this key.





# Overview

1. Null Values
2. Constraints I: General, Keys
3. Constraints II: Foreign Keys/Referential Integrity

## Foreign Keys (1)

- The relational model has no explicit relationships or pointers.
- Values for the key attributes uniquely identify a tuple.
  - They are "logical addresses" of the tuples.
- To refer to tuples of R in some other relation S, include the primary key of R among the attributes of S.
  - Such attribute values are "logical pointers" to tuples in R.
- E.g. the (Music) Piece table contains a column "CNo", and "CNo" is the primary key of "Composer".
- The join brings referring and referenced tuples together.
  - There should be exactly one matching "Composer"-tuple for every "Piece"-tuple. So the join eliminates no tuples.





## Foreign Keys (2)

CNo in Piece is a foreign key referencing Composer:

Piece

| PNo | PName | CNo |
|---|---|---|
| 1000 | The Four Seasons | 1 |
| 1002 | B-minor Mass | 2 |
| 1003 | Christmas Oratorio | 2 |
| 1004 | Missa solemnis | 3 |
| 1006 | Classical Symphony | 5 |
| 1007 | Cinderella | 5 |
| 1008 | Fireworks Music | 6 |

Composer

| CNo | CName | ... |
|---|---|---|
| 1 | Vivaldi | ... |
| 2 | Bach | ... |
| 3 | Mozart | ... |
| 4 | Dvorak | ... |
| 5 | Prokofiev | ... |

? **Error**





## Foreign Keys (3)

- The foreign key constraint means that every value for CNo appearing in Piece must also appear in the Composer table:
  $$\pi_{\mathrm{CNo}}(\mathrm{Piece}) \subseteq \pi_{\mathrm{CNo}}(\mathrm{Composer}).$$
- The values $\pi_{\mathrm{CNo}}(\mathrm{Composer})$ are a kind of "dynamic domain" for the attribute "CNo" in "Piece":
  - Only values from this set are allowed. Other values would be something like dangling pointers.
- In our simplified notation for relational schemas, we denote a foreign key by an arrow and the referenced relation:

```
Composer(CNo, CName, Born, Died)
Piece(PNo, PName, CNo → Composer)
```





## Foreign Keys (4)

- In general, a foreign key is a sequence of attributes $\{\mathrm{B}_i, \ldots, \mathrm{B}_{i+k}\}$ of a relation S, which are declared as "referencing" the primary key $\{\mathrm{A}_1, \ldots, \mathrm{A}_k\}$ of a relation R.
- This constraint requires that: $\pi_{\mathrm{B}_i,\ldots,\mathrm{B}_{i+k}}(\mathrm{S}) \subseteq \pi_{\mathrm{A}_1,\ldots,\mathrm{A}_k}(\mathrm{R})$.
  - I.e. every combination of attribute values which appears in S must also appear in R.
- The foreign key attributes can have different names than the corresponding primary key attributes.
  - The datatypes of corresponding attributes must be identical.
- In notation used here, composed foreign keys need parentheses:
  $$\mathrm{S}\big(\mathrm{B}_1, \ldots, \mathrm{B}_{i-1},\ (\mathrm{B}_i, \ldots, \mathrm{B}_{i+k}) \rightarrow \mathrm{R},\ \ldots\big).$$

## Foreign Keys (5)

- A foreign key corresponds to a one-to-many relationship in ER-diagrams. E.g. one composer can have written many pieces.
- The table “Piece” which contains the foreign key is also called the “child” table of the referential integrity constraint, and the referenced table “Composer” is the “parent table”.

  Foreign keys do not correspond to weak entities (weak entities need a foreign key because of the one-to-many relationship).
- It is possible that parent and child are the same table, e.g.

  EMP(<u>EMPNO</u>, ENAME, JOB, MGR$^o$→EMP, DEPTNO→DEPT)

  PERSON(<u>ID</u>, NAME, MOTHER$^o$→PERSON, FATHER$^o$→PERSON)





## Foreign Keys (6)

Foreign Keys and Null Values:

- Unless a “not null” constraint is explicitly specified, foreign keys can be null.
- The foreign key constraint is satisfied even if the referencing attributes are “null”. This corresponds to “nil” pointer.

  So $\mathrm{S}(\ldots, \mathrm{B}^o \rightarrow \mathrm{R}, \ldots)$ means: $\pi_{\mathrm{B}}(\mathrm{S}) \subseteq \pi_{\mathrm{A}}(\mathrm{R}) \cup \{(\mathrm{null})\}$.
- If a foreign key consists of more than one attribute, they should either all be null, or none should be null.

  But Oracle and SQL-92 allow partially defined foreign keys. In Oracle, if at least one attribute in the foreign key is null, the constraint counts as satisfied. The SQL-92 standard defines three different semantics.





## Foreign Keys (5)

Mutual References:

- Two relations can reference each other, e.g. $\mathrm{R}(\mathrm{A}, \mathrm{B}^o \rightarrow \mathrm{S})$ and $\mathrm{S}(\mathrm{B}, \mathrm{A}^o \rightarrow \mathrm{R})$.

  Such situations do happen in practice:

  EMPLOYEE(<u>EmpNo</u>, ..., Dept→DEPT)

  DEPT(<u>DNo</u>, ..., Leader$^o$→EMPLOYEE).
- Exercise/Puzzle: How can tuples be inserted?

  E.g. you want to insert tuples $(a, b)$ and $(b, a)$. The system rejects insertions which violate the constraints. Which further constraints make it impossible to fill these relations?
- Hierarchies represented in the form $\mathrm{R}(\mathrm{A}, \mathrm{B}^o \rightarrow \mathrm{R})$ are no problem: Tuples can be inserted beginning from the top.





## Important Notes

- Foreign keys are not themselves keys!

  The attributes which form a foreign key may be part of a key, but this is an exception, not the rule. The foreign key constraint has nothing to do with a key constraint.
- Only a key of a relation can be referenced, not arbitrary attributes.

  Usually, the primary key is referenced, but it is in principle possible to reference alternative keys.
- If the key of the referenced relation consists of two attributes, the foreign key must also consist of two attributes of the same two data types in the same order.

## Foreign Keys and Updates (1)

The following update operations can violate a foreign key:

- Insertion into the child table “Piece” without a matching tuple in the parent table “Composer”.
- Deletion from the parent table “Composer” where the deleted tuple is still referenced.
- Modification of the foreign key “CNo” in “Piece” to a value not in “Composer”. (Treated like an insertion.)
- Modification of the key of the parent table “Composer” where the old value is still referenced.

Deletions on “Piece” (child) and insertions on “Composer” (parent) can never lead to a violation of this foreign key constraint.





## Foreign Keys and Updates (2)

Reactions on Insertions of Dangling References:

- The insertion is rejected. The DB state remains unchanged.

Reactions on Deletions of Referenced Key Values:

- The deletion is rejected. The DB state remains unchanged.
- The deletion cascades: All tuples from Piece which reference the deleted Composer tuple are deleted, too.
- The foreign key in the referencing tuples is set to null.

  Contained in SQL-92, supported in DB2, not in Oracle.
- The foreign key in the referencing tuples is set to a declared default value.

  Contained in SQL-92, but not in Oracle or DB2.





## Foreign Keys and Updates (3)

Reactions on Updates of Referenced Key Values:

- The update is rejected. The DB state remains unchanged.

  DB2 and Oracle support only this alternative of the SQL-92 standard. In any case, changing key attributes is bad style.
- The update cascades.

  I.e. the composer number is changed in the Piece table in the same way as it was changed in the Composer table.
- The foreign key is set to null.
- The foreign key is set to a declared default value.

When specifying a foreign key, decide which reaction is best. The default is the first alternative (“No Action”).





## Exercise

Define a relational database schema for hotel.
It requires the following information:

- Information about guests: First Name, Last Name, and Home Address.
- Information about rooms: Is it single or double? What is the official room rate? When was it last renovated?
- Information about stays: Which guest stayed in which room from which date to which date? And at what room rate was he/she charged? (It might be less than the offical rate.)

Define keys, foreign keys, not null constraints. Furthermore, describe any additional constraints which might be necessary.

# Another Schema Notation

- In the Oracle DBA Exam, the following "Instance Chart" might be used to describe a relation schema:

| Instance Chart for Table PIECE | | | |
|---|---|---|---|
| Column Name: | PNO | PNAME | CNO |
| Key Type: | PK | | FK |
| Nulls/Unique: | NN, U | NN | |
| FK Table: | | | COMPOSER |
| FK Column: | | | CNO |
| Datatype: | NUMBER | VARCHAR | NUMBER |
| Length: | 4 | 40 | 2 |

- "FK" (foreign key) as a "key type" is at least misleading.