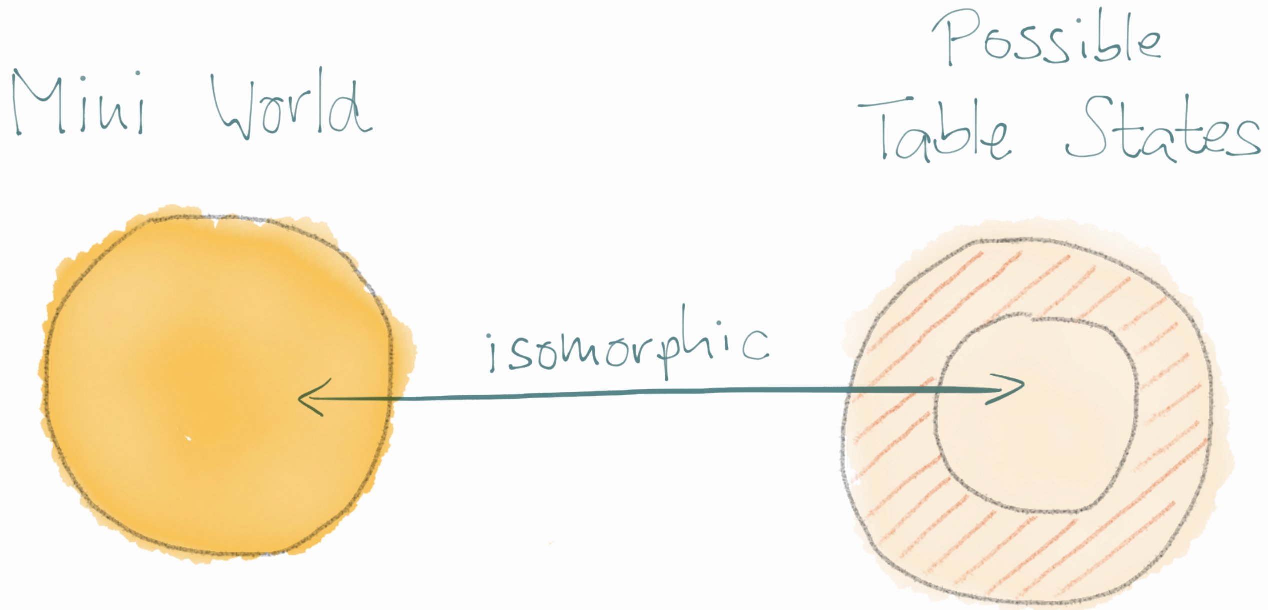
INTRODUCTION TO RELATIONAL DATABASE SYSTEMS

DATENBANKSYSTEME 1 (INF 3131)

Torsten Grust
Universität Tübingen
Winter 2021/22

CONSTRAINTS

- Schemata capture the **structural aspects** of a mini-world and largely prescribe the possible contents of a table.
- The plain definition of tables and their column types, however, often admit **too many** (meaningless, illegal) table states:



Mini world states vs. possible table states

CONSTRAINTS

calendar

no	appointment	start	stop
1	team meeting	2013-11-12 09:30	2013-11-12 10:30
2	□	2013-11-12 10:15	2013-11-12 11:45
2	lunch	2013-11-12 12:00	2013-11-12 12:30
3	presentation	2013-11-12 18:00	2013-11-12 20:00

- Mini-world rules (or: **constraints**) for table calendar (columns start and stop are of type timestamp):

1. Appointment numbers must be present and be unique.
2. Appointments may not overlap.
3. Breaks (lunches, …) should last at least one hour.
4. No appointments beyond 7pm.
5. Appointments need a defined purpose (no “catch-all” appointments).

CONSTRAINTS

Constraints

An **integrity constraint** specifies conditions which the table states have to satisfy *at all times*. This restricts the set of possible states (ideally only admits images of possible mini-world scenarios).

The **current set of constraints** $\mathbb{C}$ is integral part of the database schema:

$$(\{(R_1, \alpha_1), (R_2, \alpha_2), \ldots\}, \mathbb{C})$$

- The RDBMS will **refuse table state changes** that violate any constraint $c \in \mathbb{C}$.
- Constraints are often local (intra-column, intra-table) but may also span tables.
- Note: The attribute-type assignment $type(\cdot)$ may be understood as a set of intra-column constraints.

CONSTRAINTS

- Once a table has been created, constraints can be added to/removed from it by the SQL DDL command ALTER TABLE. In effect, ℂ is changed.

ALTER TABLE

Add to or remove constraints from existing table *t*. Various forms of actions exist:

```
ALTER TABLE [ IF EXISTS ] t
  action [, …]
```

- ALTER TABLE can also modify the schema (😦) and alter further features of a table. See the PotsgreSQL documentation. We will come back to this.

CONSTRAINTS

ALTER TABLE (continued)

action is one of the following:

```
-- column column_name may (not) hold the NULL value (□)
  ALTER column_name SET [ NOT ] NULL
-- column column_name must contain unique values
  ADD [ CONSTRAINT constraint_name ] UNIQUE (column_name [, …])
-- all rows of the table must satisfy the given condition
  ADD [ CONSTRAINT constraint_name ] CHECK (expression)
-- remove named constraint from table
  DROP CONSTRAINT [ IF EXISTS ] constraint_name
```

- Optional constraint naming allows for selective removal and dis-/enabling of constraints.

CONSTRAINTS

- ALTER TABLE notes:

1. When UNIQUE(…) contains multiple column names, the **combination of column values** must be unique in the table.
   - Constraint UNIQUE(A,B) holds for table T below, while UNIQUE(A) or UNIQUE(B) do not.
     The *more* columns are given, the *weaker* the UNIQUE constraint:

T

A	B	C
1	a	10
2	a	20
1	b	30

2. A CHECK(…) constraint **is evaluated for each table row separately** and the constraint's Boolean expression may only refer to columns of that row.

CONSTRAINTS

- Once specified, **schema and constraints** become the integral definition of the table:

```
=# \d calendar
                     Table "public.calendar"
+-------------+-----------------------------+-----------+----------+---------+
|   Column    |            Type             | Collation | Nullable | Default |
+-------------+-----------------------------+-----------+----------+---------+
| no          | integer                     |           | not null |         |
| appointment | text                        |           | not null |         |
| start       | timestamp without time zone |           |          |         |
| stop        | timestamp without time zone |           |          |         |
+-------------+-----------------------------+-----------+----------+---------+
Indexes:
    "calendar_no_key" UNIQUE CONSTRAINT, btree (no)
Check constraints:
    "start before stop" CHECK (start < stop)
    "breaks last longer than one hour" CHECK ((appointment <> ALL (ARRAY['lunch',
'break']))
                                              OR (stop - start) >=
'01:00:00'::interval)
    "no appointments beyond 7pm" CHECK (stop::time <= '19:00:00'::time)
```

KEY CONSTRAINTS

- UNIQUE constraints are of particular importance for the relational data model. Since **rows cannot be addressed by memory location or table position**, we need to

> **identify** individual **rows** of a table **by value** (of selected columns).

- Value-based row identification is so important, that RDBMS automatically create auxiliary data structures (e.g. B-tree **indexes**, see line marked **A** on previous slide → *DB 2*) to efficiently:

1. check that uniqueness is not violated when rows are added/modified, and
2. **find a row given its unique column values.**

- Recall: Uniqueness of the piece ID in tables bricks.cvs and minifigs.csv helped us to optimize the *weight of LEGO Set 5610* PyQL queries.

KEY CONSTRAINTS

Key

A **key** of a table $R(a_1, \ldots, a_n)$ is a set of columns $K \subseteq \{a_1, \ldots, a_n\}$ that **uniquely identifies** the rows of R:

$$\forall\ t, u \in inst(R): t.K = u.K \Rightarrow t = u$$

Read: "*If two rows agree on the columns in K, they are indeed the same row.*"
Or: "*The values of the columns in K already suffice to locate a row in R.*"

- Note: Here we have generalized dot notation to work over column sets (not just single columns). If $K = \{a_{i1}, \ldots, a_{ik}\}$, then $t.K = (t.a_{i1}, \ldots, t.a_{ik})$. As before, equality of row values (via =) is defined column by column.

KEY CONSTRAINTS

Quiz: Keys for Tables?

R

A	B	C
1	true	30
2	false	10
3	true	20

S

A	B	C
1	true	20
1	true	10
2	false	10

T

A	B	C
1	true	30
2	false	10
1	true	30

KEY CONSTRAINTS

- Notes:

1. While (SQL) tables may have **no key** at all, this is impossible for relations in the sense of the original relational model. (Why?)
2. If K is key for $R(a_1, \ldots, a_n)$, then any K^+ with $K \subseteq K^+ \subseteq \{a_1, \ldots, a_n\}$ also is a key for R. Such keys K^+ are also referred to as **superkeys**.
3. Keys of **minimal size** (minimum column count) are the so-called **candidate keys** of R.
   Key K is minimal if each attribute $a \in K$ is essential for row identification: let $K^- = K - \{a\}$, then
   $$\exists\ t, u \in inst(R)\colon t.K^- = u.K^- \land t \neq u$$
4. Keys are constraints: they have to be satisfied in **all table states**, not only the current one.

KEY CONSTRAINTS

Example: Keys for the LEGO Set Mini-World

contains (excerpt)

set	piece	color	extra	quantity
00-1	29c01	1	f	6
00-1	29c01	5	f	8
00-1	3001old	5	f	9
00-4	3001old	3	f	4
00-4	3001old	3	t	1

- One LEGO set contains multiple pieces: {set} cannot be key. One piece occurs in different sets: {piece} cannot be key.
- In a set, a piece may occur multiple times (in various colors): {set,piece} is no key.
- From the excerpt above, {quantity} could be key, but knowledge of the mini-world says that this is coincidence (will *not* hold for all table states).
- {set,piece,color,extra} is **candidate key** for table contains (“*Provide set number, LEGO piece ID, color, extra status to identify the piece’s row.*”)

KEY CONSTRAINTS

Example: Keys for the LEGO Set Mini-World

bricks

piece	type	name	cat	weight	img	x	y	z
08010ac01	B	Electric, Light Brick 12V …	123	2.79999995	http://www…/PL/08010ac01	2	2	1
08010bc02	B	Electric, Light Brick 12V …	123	2.79999995	http://www…/PL/08010bc02	2	2	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

minifigs

piece	type	name	cat	weight	img
sw038	M	Watto	65	5.32999992	http://www…/ML/sw038
sw040	M	Royal Guard	65	4.1500001	http://www…/ML/sw040
85863pb099	M	Microfig Legends of Chima Eagle	252	□	http://www…/ML/85863pb099
⋮	⋮	⋮	⋮	⋮	

- {piece} and {img} are **candidate keys** in both tables (we would need to scan the entire table to be sure — but mini-world knowledge suggests so).
- In table minifigs, {weight} could be candidate key. Mini-world knowledge says otherwise. Also, presence of □ (i.e., SQL NULL) is problematic: is NULL = NULL?

PRIMARY KEY

Primary Key
Among the *candidate keys* of table R, one key K is selected to be the **primary key.** It is expected that users/applications will identify rows of R primarily based on K.

- Selection of primary key K is primarily driven by

1. **Pragmatics** ("*this is how things are naturally identified in this mini-world*")
2. **Efficiency**
   Can expect frequent evaluation of predicates of the form $t.K = u.K$ (for rows t, u) in queries. Minimum key size helps but equality comparison on key components should be efficient as well.
   (Cf. candidate keys {piece} vs. {img} on previous slide.)

PRIMARY KEY

ALTER TABLE … PRIMARY KEY

The SQL DDL command

```
ALTER TABLE [ IF EXISTS ] t
  ADD PRIMARY KEY (column_name [, …])
```

establishes the specified columns as *the* **primary key** of table *t* (there can be only one primary key).

- Primary keys serve as *the* row identifier and thus *must* be present and unique. Constraint PRIMARY KEY (a_1,…,a_k) implies the following k+1 constraints:

1. a_1 NOT NULL, ..., a_k NOT NULL, and
2. UNIQUE (a_1,…,a_k).

PRIMARY KEY (NOTATION)

- In SQL table design, it is *customary* to place the primary key column(s) first ("left") in the schema.
- In schemata and illustrations of keyed tables, key columns typically are underlined or otherwise emphasized:

$$R(\underline{A}, \underline{B}, C, D)$$

R