Data dictionaries and specifications

A shared description of data

A data dictionary or specification documents what each data item means and how it must be represented. It helps user interfaces, algorithms, databases and exchanged messages agree.

> **Key idea:** A field name and type are a start, not a complete specification. Meaning, format, requirement, allowed values and examples remove ambiguity.

Field	Type	Required	Rule	Example
`booking_id`	INTEGER	Yes	Unique; positive; generated by system	`1842`
`status`	VARCHAR	Yes	One of pending, confirmed, cancelled	`confirmed`

Choose data types by meaning and use

Type	Suitable for	Example
INTEGER	Whole-number quantities and generated identifiers	`capacity = 30`
DECIMAL	Values requiring a fractional part	`price = 12.50`

Type	Suitable for	Example
VARCHAR	Variable-length text and identifier-like digits not used in arithmetic	`postcode = "0400"`
BOOLEAN	Two-state values	`accessible = true`
DATE / DATETIME	Dates and timestamps that must be sorted or compared	`2026-08-05T14:30:00+10:00`

> **Digits do not guarantee a numeric type:** phone numbers, postcodes and licence numbers are identifiers. Arithmetic is not meaningful, and leading zeroes may matter.

Choose types

Determine suitable types for `student_name`, `attendee_count`, `paid`, `start_time` and `phone_number`.

Check your answer

VARCHAR, INTEGER, BOOLEAN, DATETIME and VARCHAR respectively.

Names should reveal meaning

- Use one concept per field: `first_name` and `family_name` if they must be processed separately.
- Include units when otherwise ambiguous: `duration_minutes`.
- Use consistent conventions: do not mix `eventID`, `event_id` and `EventId`.
- Use stable identifiers: `venue_id` should not be replaced by a changeable venue name.
- Avoid vague fields such as `data`, `value` or `type` without context.

Improve the names

A sensor table contains `id`, `value`, `time` and `ok`. The value is temperature in degrees Celsius, and `ok` records whether calibration is valid.

Recommend clearer field names.

View a sample solution

`sensor_id`, `temperature_celsius`, `recorded_at` and `calibration_valid`.

Constraints and validation rules

A constraint limits stored values. Validation checks input against those rules.

- **Required:** a value must be supplied.
- **Range:** a number or date lies within stated limits.
- **Length:** text has an allowed size.
- **Format:** data follows an agreed pattern.
- **Allowed values:** a value belongs to an enumerated set.
- **Uniqueness:** no other record uses the same value.
- **Referential:** a foreign identifier matches an existing related record.

Validation checks plausibility and rule compliance; it does not prove truth. A correctly formatted but mistyped email may pass validation.

Validation and verification

Validation tests data against rules. **Verification** checks that data was transferred or entered as intended, such as comparing an entry with its source.

A date picker can validate that a booking is not in the past. A confirmation screen can help the user verify that the selected venue and date match their intention.

Classify each check

1. An email must contain a non-empty local part and domain.
2. A staff member compares a scanned consent form with the entered record.
3. A password is entered twice and the values must match.

Check your answer

1. Validation: format rule.
2. Verification: comparison with the source.

3. Verification: matching entries reduces transcription error, although it does not prove the chosen password is appropriate.

Keys and relationships

A **primary key** uniquely identifies a record in its table. A **foreign key** stores the primary-key value of a related record.

```
VENUE(venue_id, venue_name, suburb)
BOOKING(booking_id, venue_id, booking_date, attendee_count)
```

`VENUE.venue_id` identifies a venue. `BOOKING.venue_id` links each booking to an existing venue without repeating the venue's name and suburb in every booking.

> **Referential integrity:** a booking should not contain a `venue_id` that does not exist in VENUE.

Explain the relationship

Explain why `venue_id` is preferable to storing only `venue_name` in BOOKING.

Check your answer

A stable identifier links the booking to one exact venue even if names are duplicated or later changed. Venue details remain in one record, reducing repeated data and inconsistent updates.

Specifications across a solution

The same rule should remain compatible across UI, JSON, algorithms and database storage.

Component	Implementation of `status`

UI	Select list containing pending, confirmed and cancelled
JSON	`"status": "confirmed"`
Validation	Reject values outside the allowed set
Database	Non-null text field with an allowed-value constraint

Evaluate fitness for purpose

A specification is useful only if it stores the data required to answer questions and perform processes.

A dataset has `region`, `licence_type`, `valid_licence` and `registration_date`. It must track how one licence changes between active, suspended and expired over time.

Evaluate the specification and **Recommend** a refinement.

View a sample solution

The Boolean `valid_licence` cannot distinguish active, suspended and expired, and no stable licence identifier connects records over time. Add `licence_id` as a unique VARCHAR and replace the Boolean with `licence_status`, restricted to the required status values. This allows changes for the same licence to be grouped and compared.

Independent practice: equipment loans

Design a data dictionary for a school equipment-loan record. The system must identify the loan, link it to an existing student and item, record checkout and optional return times, store item condition as *good*, *damaged* or *missing*, and record whether staff have checked the return.

Determine a field name, data type, required state, constraint and example for each item. Then **Justify** the types used for the student identifier and checked state.

View a sample solution

Field	Type	Required	Rule	Example
`loan_id`	INTEGER	Yes	Unique, positive	408
`student_id`	VARCHAR	Yes	Existing student	S00421
`item_id`	VARCHAR	Yes	Existing item	CAM-12
`checked_out_at`	DATETIME	Yes	Valid timestamp	2026-08-05T09:20:00+10:00
`returned_at`	DATETIME	No	Null or not before checkout	null
`condition`	VARCHAR	Yes	good, damaged or missing	good
`return_checked`	BOOLEAN	Yes	true or false	false

`student_id` is VARCHAR because it contains a letter and identifies rather than measures.
`return_checked` is Boolean because it has exactly two states.

Revision #5
Created 2026-10-06 00:40:27 UTC by Mr Napper
Updated 2026-10-06 01:32:10 UTC by Mr Napper