# Database Schemas Revisited

The schema defines the relational model for a database; how data is divided into tables in Database design has an impact on storage requirements and efficiency of accessing data.

Good schema design delivers:

- Minimal redundancy of information
  - The same information should not appear in multiple places
- Easy to understand the relationship of the data
  - Information is properly organised or split into logical pieces it easy to access.
- Database performs fast and efficiently
  - Data is organised to find information, with fewer steps, and without retrieving more data than needed.

Important because most web applications are transaction intensive, i.e., users are often creating, reading, updating, and deleting data.

# Adapting our Schema to SQL

# Ensuring Relational Data Integrity

# Integrity Constraints

## What are they?

Conditions put in place on the database to:

- Guard against accidental damage to the database
- And ensure that changes don't result in a loss of consistency in the data
- Restrict valid values for data
- Ensure that related data is deleted from all tables

Examples

- Key declaration: Definition of primary keys means that updates are constrained
- Mapping cardinality: Constrains the set of relationships
- Value constraints: Ensure that values in a given column are restricted to meet specific conditions

# Primary & Composite Keys

- Specify a single primary key or composite key after the column names
  - Primary Key: `PRIMARY KEY (column_name)`
  - Composite Key: `PRIMARY KEY (column1_name,column2_name)`

```
CREATE TABLE Customer (
    id INT,
    name VARCHAR(30),
    street VARCHAR(50),
    suburb VARCHAR(30),
    PRIMARY KEY (id)
);
```

# Foreign Keys

- In relational databases, foreign keys are used to establish a relationship between two tables
- A foreign key is a set of attributes that refer to a primary key of another table that the table has a relationship with.
- Specify one or more forign keys after the primary/composite key
  - Single Foreign Key: `FOREIGN KEY (column_name) REFERENCES RelationshipTable(column_name)`
  - Each foreign key needs its own declaration

```
CREATE TABLE Order (
    order_id INT,
    cust_id INT,
    PRIMARY KEY (order_id),
    FOREIGN KEY (cust_id) REFERENCES Customer(cust_id)
);
```

# Referential Integrity

Ensures that values that appear in one table for a given set of attributes also appear for a certain set of attributes in another table

## Example:

If a customer has several bank accounts, then deleting the customer from the system should also delete their accounts.

## TL;DR:

- What requirements do our realtionships have?
- What happens to related data when I delete something?

# Referential Integrity in SQL; Declaring Foreign Keys

Referential integrity in SQL is built upon the relationships defined using **foreign keys**

```
CREATE TABLE Customer (
    id INT,
    name VARCHAR(30),
    PRIMARY KEY (id)
);
```

We can declare a foreign key using the `FOREIGN KEY` constraint:

```
CREATE TABLE Account (
    number INT,
    cust_id INT,
    name VARCHAR(30),
    open_date DATE,
    PRIMARY KEY (number),
    FOREIGN KEY (cust_id) REFERENCES Customer(id)
);
```

Depending on the relationship, we can select actions to occur when an entity involved in some relationship is deleted.

# Referential Integrity on Delete in SQL

```
CREATE TABLE Customer (
    id INT,
    name VARCHAR(30),
    PRIMARY KEY (id)
);
```

## Set NULL

- Use when the relationship is *optional*; Not all rows in the FK table are involved in the relationship
- Use the `ON DELETE SET NULL` clause to indicate that when the main entity is deleted, the value of the foreign key in the associated entity is set to NULL

```
CREATE TABLE Account (
    number INT,
    cust_id INT,
    name VARCHAR(30),
    open_date DATE,
    PRIMARY KEY (number),
    FOREIGN KEY (cust_id) REFERENCES Customer(id) ON DELETE SET NULL
);
```

# Referential Integrity on Delete in SQL

```sql
CREATE TABLE Product (
    id INT,
    name varchar(30),
    PRIMARY KEY (id)
);
```

## No Action

- Use when the relationship is *required*, but the row's existence **not** defined by the relationship
  - Example: All ordered items have to be contained in orders
- Use the `ON DELETE NO ACTION` clause to indicate that the main entity cannot be deleted if it has an association with another entity.

```sql
CREATE TABLE Inventory (
    inventory_id INT,
    product_id INT NOT NULL,
    quantity INT,
    PRIMARY KEY (inventory_id),
    FOREIGN KEY (product_id) REFERENCES Product(id) ON DELETE NO ACTION
);
```

# Referential Integrity in SQL

```
CREATE TABLE Order (
    id INT,
    date DATE,
    PRIMARY KEY (id)
);
```

## Cascade

- Use when the relationship is *required*, and the row's existence is defined by the relationship
  - E.g. Foriegn Key is also part of the table's Primary Key
- Use the `ON DELETE CASCADE` clause to indicate that when the main entity is deleted, all associated entities for that Foreign Key will also be deleted.

```
CREATE TABLE OrderedItem (
    order_id INT,
    item_id INT,
    quantity INT,
    PRIMARY KEY (order_id,item_id),
    FOREIGN KEY (order_id) REFERENCES Order(id) ON DELETE CASCADE
);
```

# Value Integrity

Ensures that values for a given column are valid

## Example:

A customer's shopping cart cannot have fewer than 0 (negative) items.

Value integrity in SQL is achieved using a number of additional clauses when creating tables

# Value Integrity in SQL

## The CHECK Clause

The check clause permits attributes to be restricted to a predefined set or series of values:

```
CREATE TABLE Persons (
    id INT,
    LastName VARCHAR(255),
    FirstName VARCHAR(255),
    Age INT,
    City VARCHAR(255),
    CHECK (Age>=18 AND City='Adelaide')
);
```

We can also name constraints

- Use `CONSTRAINT name`
- Useful to indicate which constraint an update violated:

```
    CONSTRAINT UnderAge CHECK (Age>=18)
```

# Value Integrity in SQL

## Other constraints

`UNIQUE`

- Values in this column are all different; No duplicates

`NOT NULL`

- Values in this column can not be `NULL`

`DEFAULT`

- Sets a default value

# Value Integrity in SQL

## Other constraints

`AUTO_INCREMENT`

- Is used with numeric data to set the value to a number that increases by 1 for each new row.
- We don't need to specify these columns in `INSERT` queries.

## Example:

```
CREATE TABLE Account (
    number INT UNIQUE AUTO_INCREMENT,
    branch_name VARCHAR(30) NOT NULL,
    balance number(5) DEFAULT(0)
);
```

Fun fact: `PRIMARY KEY` is equivalent to the combination of both `NOT NULL` and `UNIQUE`

# Summary

- Data Integrity can be enforced in our RDBMS.
- Referential Integrity clauses ensure that relationships between entities are upheld.
- Value Integrity clauses ensure that invalid data cannot be entered into the database.