Database

SQL

Creating and Populating Tables with Constraints

Case Study

Game Store

Requirement

Game Store Requirement

Our company, **Apasaja Pte Ltd**, has been commissioned to develop an application to manage the data of an online app store. We want to store several items of information about our customers such as their **first name**, **last name**, **date of birth**, **e-mail**, **date** and **country of registration** to our online sales service and the **customer identifier** that they have chosen.

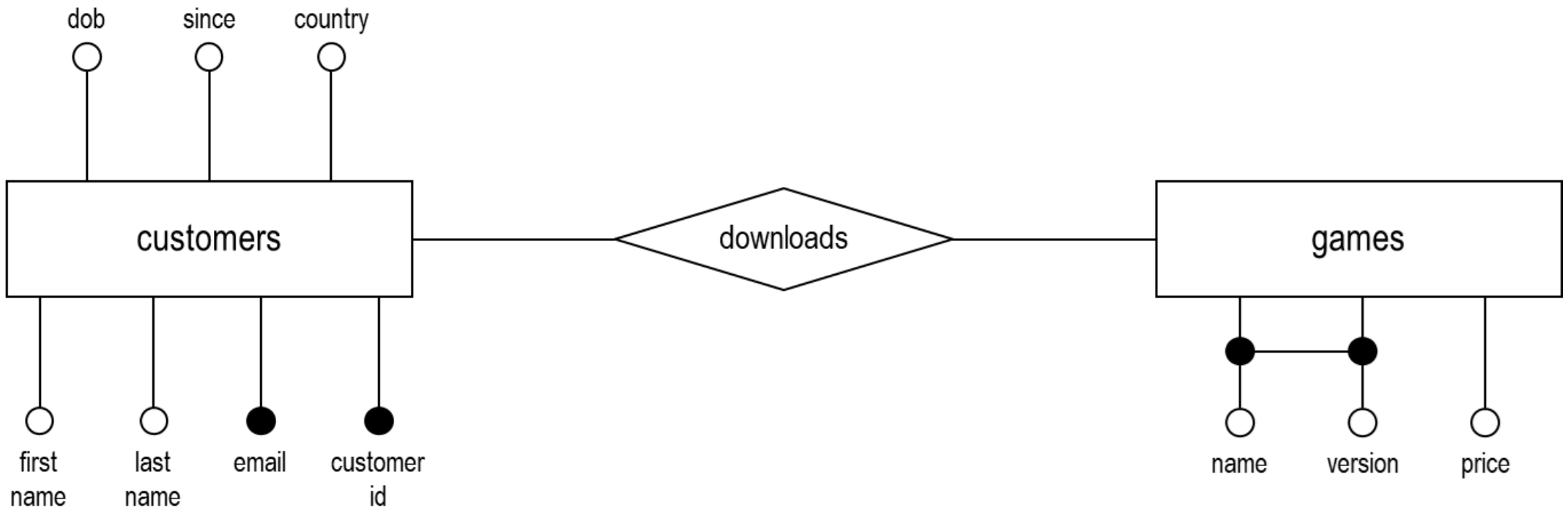
We also want to manage the list of our products, games, their **name**, their **version**, and their **price**. The price is fixed for each version of each game. Finally, our customers buy and download games. We record which version of which game each customer has downloaded. It is not essential to keep the download date for this application.

*Download from Canvas "**Files > Cases > AppStore**"

Case Study

Design

Entity-Relationship Diagram



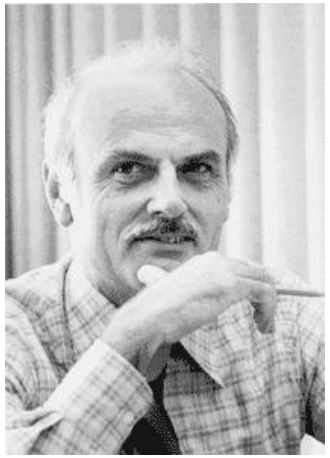
RDBMS

Database Management Systems

Definition

Most popular database management systems since the 1970's are relational. They implement the **relational model** of **Edgar F. Codd**, a simple and rigourous **data model** based on **relations** implemented as **tables**.

They support a language for the **definition**, **manipulation**, **query**, and **control**, which intersects the international standard for the **SQL** language.

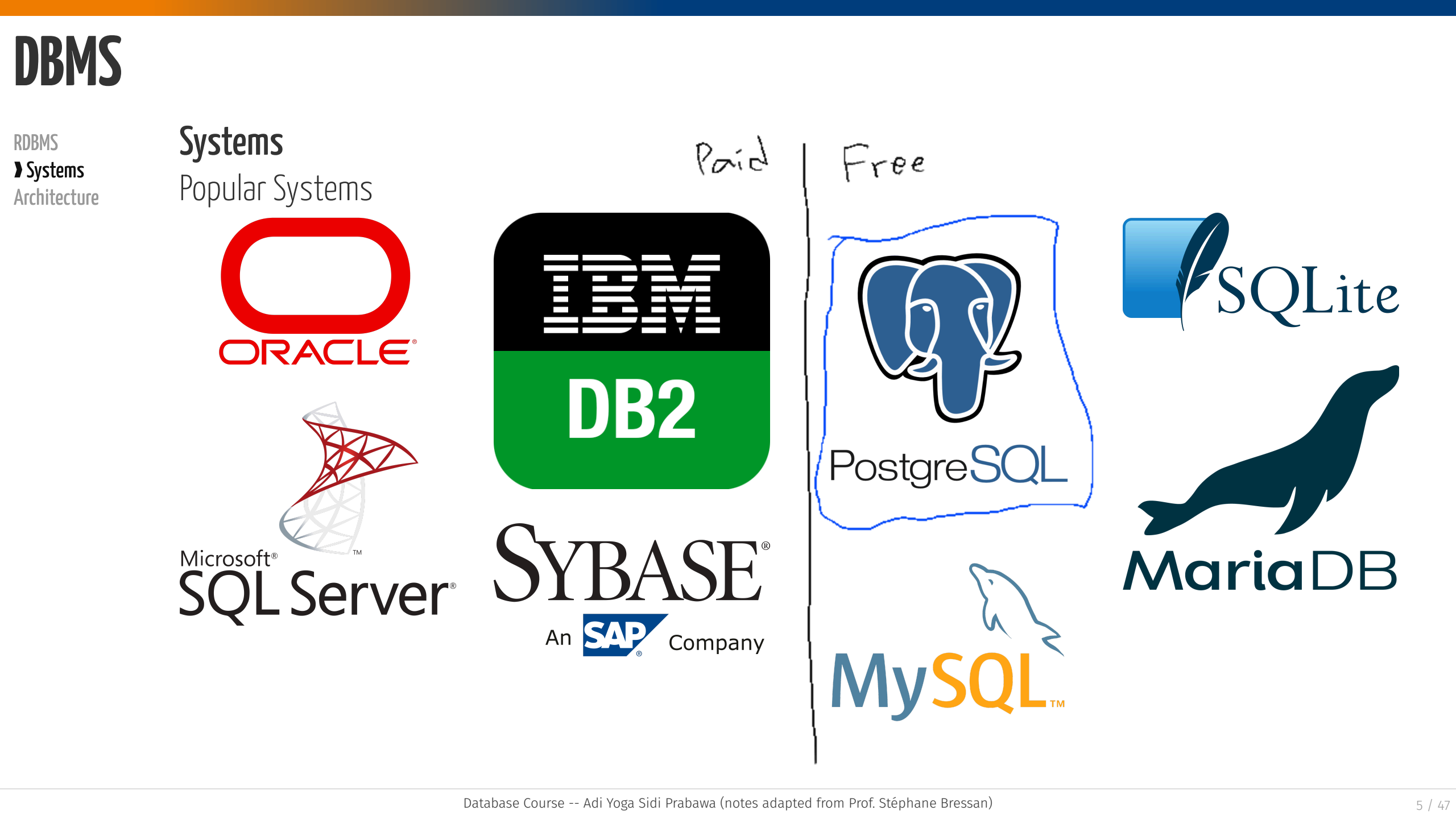


> "*Future users of large data banks must be protected from having to know how the data is organized in the machine.*"
>
> Edgar F. Codd
> "A Relational Model for Large Shared Data Banks"

DBMS

Systems

Popular Systems

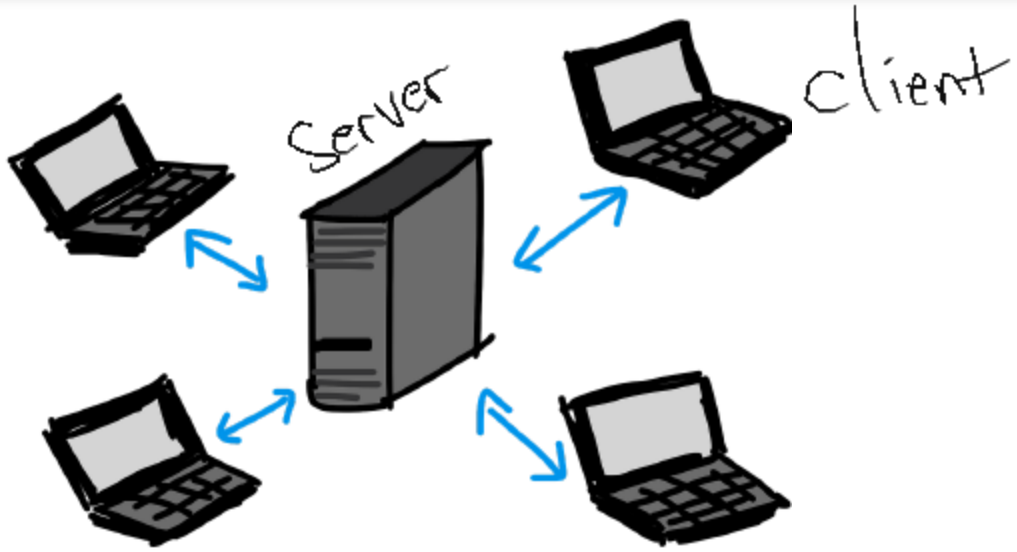


Architecture

Client-Server

Client-Server Architecture

Most popular relational database management systems since the 1970's have a **client-server** architecture and manage and control distributed and concurrent access to, usually large amounts of, persistent data on secondary storage, persistent data.

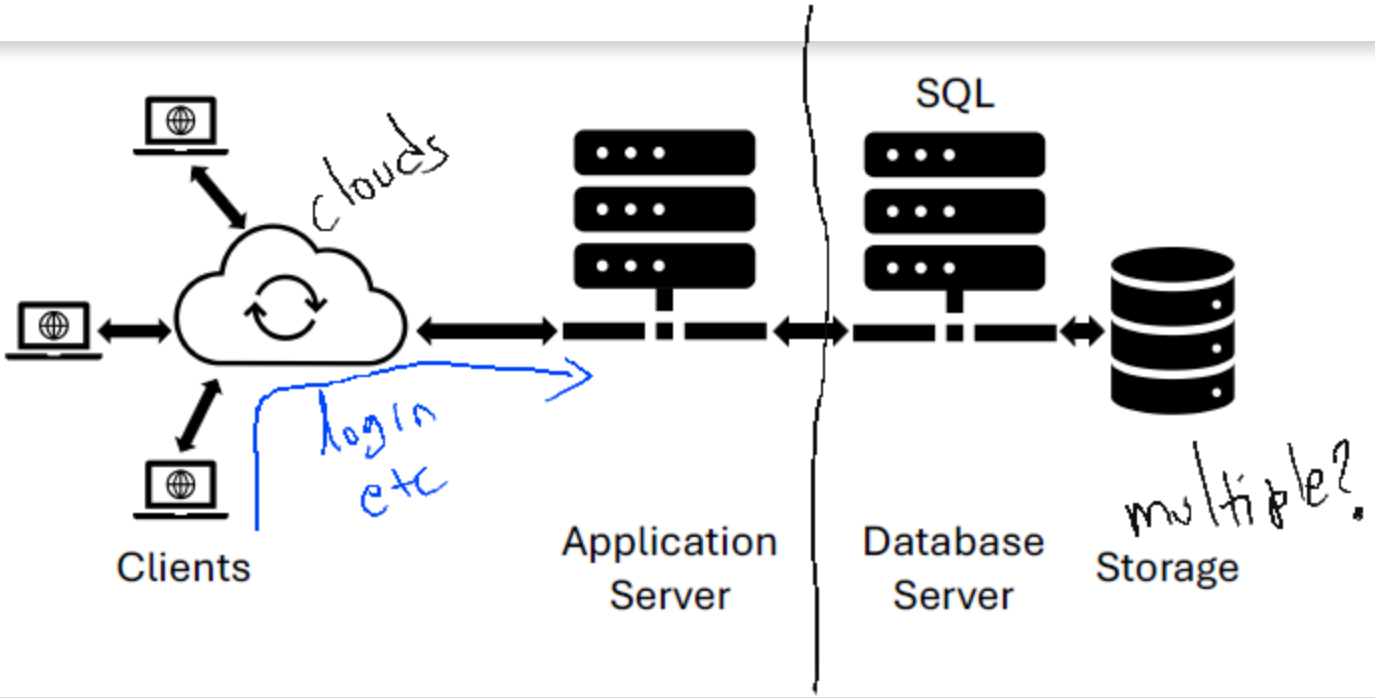


Architecture

Three-Tier

Three-Tier Architecture

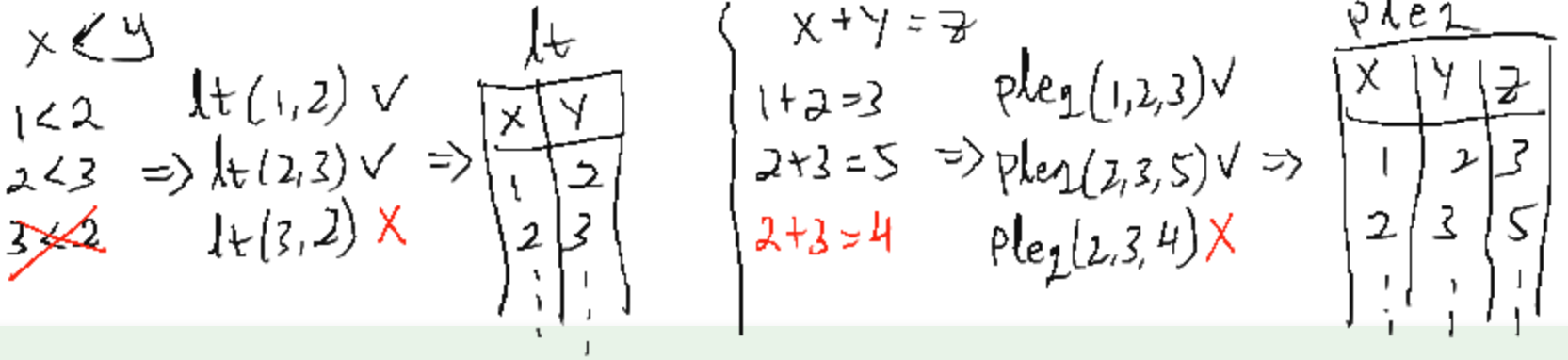
Most modern database applications have a **three-tier** architecture. A three-tier architecture is a client-server software architecture that divides applications logically and physically into three distinct tiers: the presentation tier or *clients*, the business logic tier or *application server*, and the data tier or *database server*.



Logical Design

Model

Relational



Relation in Theory

In theory, the **relational model** proposes to organise data in relations. Relations have a name and attributes. Attributes have a name.

relates values

Mathematically, relations are subsets of the Cartesian product of the domains of their attributes. Domains are extensions of types.

header

first_name	last_name	email	dob	since	customerid	country
Johnny	Gilbert	jgilberte8@nymag.com	1995-06-19	2023-07-15	JohnnyG89	Malaysia
Deborah	Ruiz	druiz0@drupal.org	1990-08-01	2024-04-17	Deborah84	Singapore
Tammy	Lee	tlee1@barnesandnoble.com	2004-09-14	2024-02-21	Tammy1998	Singapore
...						

values are related

* only stores true values

Logical Design

Model

Relational

Relation in Practice

In practice, the **relational database management systems** organise data in **tables**. Tables have a **name**. Tables are **multi-sets** *(not lists/sets)* of **rows** or **records**. Rows or records have **fields** corresponding to the **columns** of the table.

Columns or fields have a **name**. Columns or fields also have an **implicit position** indicated by the order in the corresponding creation statement. Columns or fields have a **domain** which is a type to the extension of which is added the possibility of a `NULL` value.

We shall discuss in a subsequent lecture the syntax, semantics, and behavior of `NULL` values and of the constructs that are used to manipulate them.

Logical Design

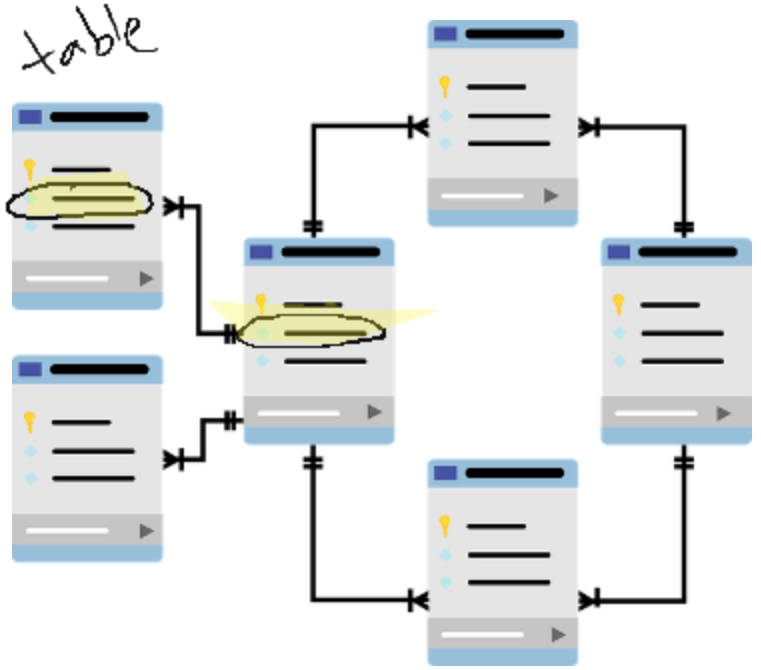
Design

Logical

Database Logical Design

Logical design is the activity consisting in choosing the appropriate schema for the database application.

It consists mainly the number of tables, the names of the tables, the number of columns each table should have and the names and domains of the columns.

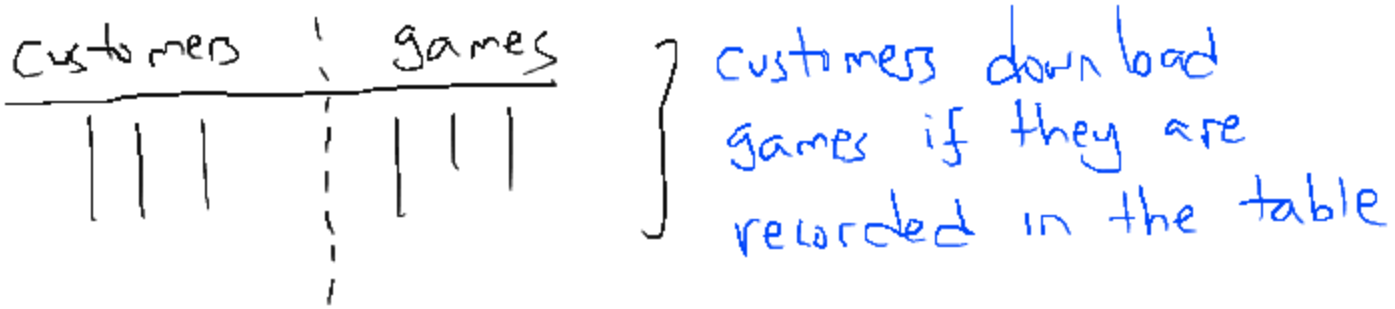


Logical Design

Creation

Basic

CREATE TABLE




```
CREATE TABLE downloads (
  first_name   VARCHAR(64),
  last_name    VARCHAR(64),
  email        VARCHAR(64),
  dob          DATE,
  since        DATE,
  customerid   VARCHAR(16),
  country      VARCHAR(16),
  name         VARCHAR(32),
  version      CHAR(3),
  price        NUMERIC
);
```

table name

customers

games

column name

type

no comma at the end

format : column_name type,

Note

The choice of the number of columns and of their domains implicitly imposes **structural constraints**.

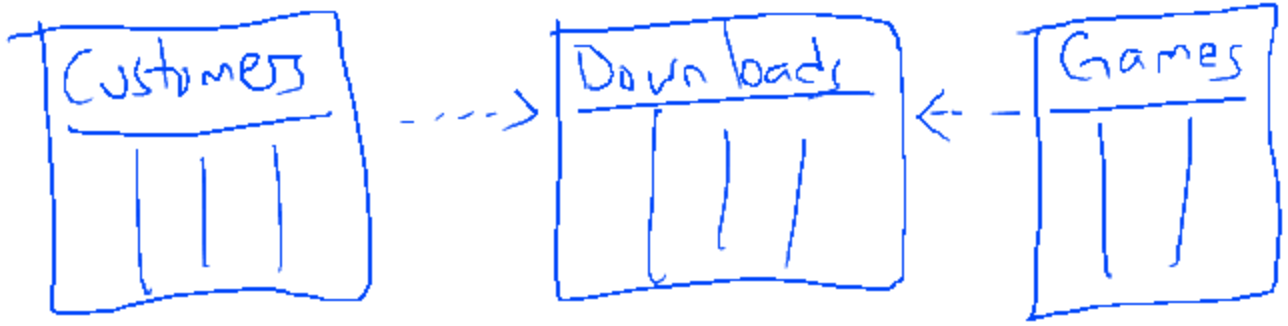
For instance, if we use only **one table**, there can be **no customer without a game** and **no game without a customer**, unless we use NULL values.

We can use the structural constraints as integrity constraints to control and maintain the **integrity of data** because data that does not fit in the table cannot be stored and only data that fit can be stored.

Logical Design

Creation

Decomposed Table Creation




```
CREATE TABLE customers (
  first_name   VARCHAR(64),
  last_name    VARCHAR(64),
  email        VARCHAR(64),
  dob          DATE,
  since        DATE,
  customerid   VARCHAR(16),
  country      VARCHAR(16)
);
```

```
CREATE TABLE games (
  name         VARCHAR(32),
  version      CHAR(3),
  price        NUMERIC
);
CREATE TABLE downloads (
  customerid   VARCHAR(16),
  name         VARCHAR(32),
  version      CHAR(3)
);
```

We opt for a **schema** with three tables above with the indicated **columns** and their respective **domains** *(e.g., `VARCHAR(64)`, `DATE`, `CHAR(3)`, `NUMERIC`, etc)**.

*See: https://www.postgresql.org/docs/current/datatype.html

Logical Design

Creation

Decomposed Table Creation

```
CREATE TABLE customers (
  first_name  VARCHAR(64),
  last_name   VARCHAR(64),
  email       VARCHAR(64),
  dob         DATE,
  since       DATE,
  customerid  VARCHAR(16),
  country     VARCHAR(16)
);
```

Note

Looking only at the `customers` table, we can add a new customer even if the customer has **not downloaded** any games.

We do not have to introduce `NULL` values.

Logical Design

Creation

Decomposed Table Creation

Note

Similarly for the **games** table, we can add a new game even if there are **no customer** that has downloaded the game.

We do not have to introduce **NULL** values.

```
CREATE TABLE games (
  name          VARCHAR(32),
  version       CHAR(3),
  price         NUMERIC
);
CREATE TABLE downloads (
  customerid    VARCHAR(16),
  name          VARCHAR(32),
  version       CHAR(3)
);
```

Logical Design

Creation

Decomposed Table Creation

Note

Unlike using a single table, by splitting the table, we only need to remember the **identifier** of `customers` *(i.e., `customerid`)* and `games` *(i.e., `(name, version)`)*.

As customers *(resp. games)* can be identified by `customerid` *(resp. `(name, version)`)*, we can obtain the rest of the information if we want.

This uses **query** which is the topic for next week.

```
CREATE TABLE games (
  name        VARCHAR(32),
  version     CHAR(3),
  price       NUMERIC
);
CREATE TABLE downloads (
  customerid  VARCHAR(16),
  name        VARCHAR(32),
  version     CHAR(3)
);
```

Break

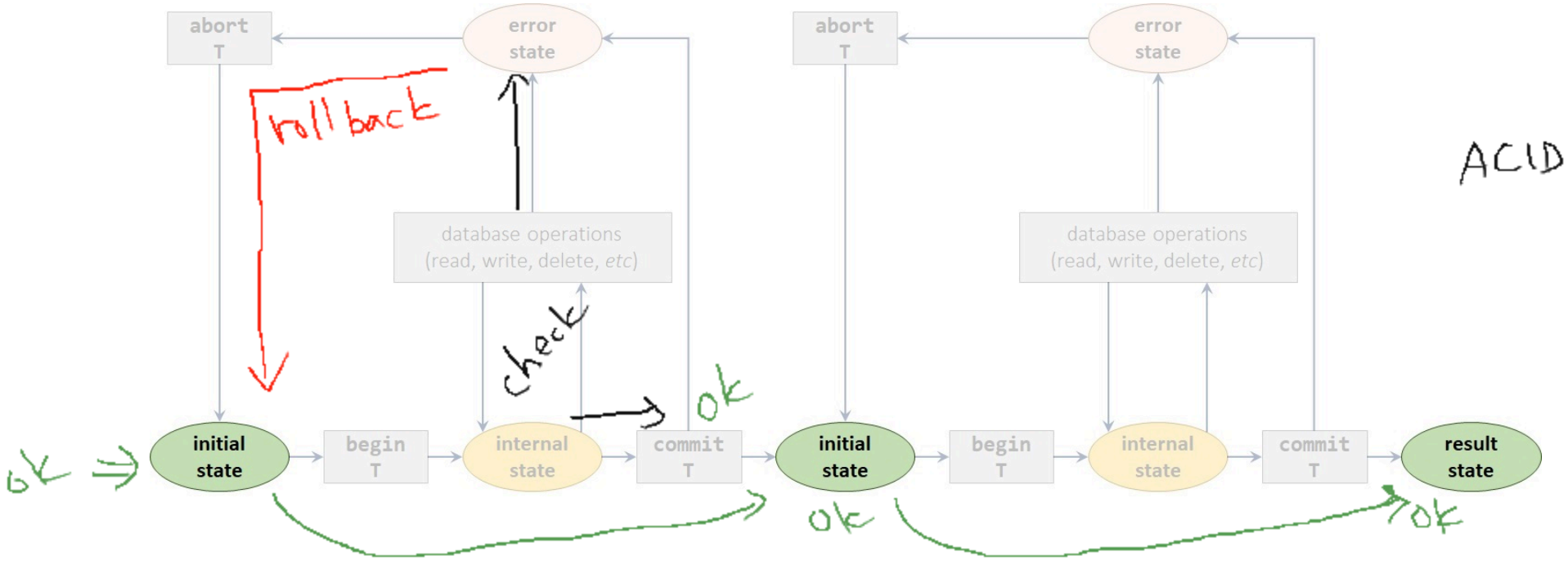
Constraints Check

Constraints

Integrity

Integrity Constraints

A **consistent state** of the database is a state which complies with the business rules as defined by the structural constraints and the integrity constraints in the schema.



Properties

ACID
- Atomicity
- Consistency
- Isolation
- Durability

Constraints Check

Constraints

Integrity

Integrity Constraints

A **consistent state** of the database is a state which complies with the business rules as defined by the **structural constraints** and the **integrity constraints** in the schema.

SQL allows to express business rules, such as "*students who have not passed cs1010 cannot take cs2020*" as integrity constraints.

If an integrity constraint is **violated** by an **operation** *(i.e., an insertion, update or deletion)*, or a **transaction** *(i.e., a sequence of operations)*, the **operation** or the **operation** is **aborted** and **rolled back** and its changes are undone, otherwise, it is **committed** and its changes are effective for all users.

The database management system concurrency control and recovery mechanisms should ensure **atomicity**, **consistency**, **isolation**, and **durability** of transactions *(i.e., ACID properties)*.

Constraints Check

Constraints

in Practice

Transaction

In practice, there are different ways to specify the scope of transactions in SQL or application code. One of them is to use blocks with keywords such as `BEGIN`, `END` or `COMMIT`, `ABORT` and `ROLLBACK`.

Deferred

In most systems, unfortunately, integrity constraints are **immediate** *(i.e. they are checked after each operation)*. It is preferable to set all integrity constraints to be **deferred** *(i.e. they are checked after the end of transactions)*. Unfortunately again, most systems, like PostgreSQL, do only allow certain constraints to be deferred.

Constraints Check

Constraints

Support

Integrity Constraints

SQL Support five kinds of integrity constraints:

```
NOT NULL
PRIMARY KEY
UNIQUE
FOREIGN KEY
CHECK
```

Constraints Check

Recreation

Delete

DELETE FROM

```
DELETE FROM customers;
```

DELETE deletes the **content** of the table but **NOT** its definition.

DROP TABLE

```
DROP TABLE customers;
```

DROP deletes the **content** and the **definition** of the table.

*We recommend **W3Schools** for details of syntax and behavior of standard SQL construct.
For PostgreSQL specific behavior, see **PostgreSQL Documentation** instead.

Constraints Check

Recreation

Primary Key

Definition

A **primary key** is a set of columns that uniquely identifies a record in the table. Each table has at most one primary key.

Note

The primary key can be **one column** or a **combination of columns**. An instance of the table cannot have two records with the same value or combination of values in the primary key columns and no primary key column cannot contain a NULL value*.

*This is not implemented in all database management systems.

Constraints Check

Recreation

Primary Key

Column Constraint

```
CREATE TABLE customers (
  first_name   VARCHAR(64),
  last_name    VARCHAR(64),
  email        VARCHAR(64),
  dob          DATE,
  since        DATE,
  customerid   VARCHAR(16)
  PRIMARY KEY, -- applies to the row
  country      VARCHAR(16)
);
```

Table Constraint

```
CREATE TABLE customers (
  first_name   VARCHAR(64),
  last_name    VARCHAR(64),
  email        VARCHAR(64),
  dob          DATE,
  since        DATE,
  customerid   VARCHAR(16),
  country      VARCHAR(16),
  PRIMARY KEY (customerid) -- after row
);
```

customers(first_name, last_name, email, dob, since, customerid, country)

Constraints Check

Recreation

Primary Key

Definition

```
CREATE TABLE customers (
  first_name   VARCHAR(64),
  last_name    VARCHAR(64),
  email        VARCHAR(64),
  dob          DATE,
  since        DATE,
  customerid   VARCHAR(16),
  country      VARCHAR(16),
  PRIMARY KEY customerid -- after row
);
```

INSERT INTO

```
INSERT INTO customers VALUES (
  'Carole',
  'Yoga',
  'cyoga@glarge.org',
  '2002-08-01',
  '2024-08-09',
  'Carole89',
  'France'
);
-- INSERT INTO tries to insert a row
```

Error

Try inserting another one with the same `customerid`, we will get an error.

Constraints Check

Recreation

Primary Key

Composite

```
CREATE TABLE games (
  name          VARCHAR(32),
  version       CHAR(3),
  price         NUMERIC,
  PRIMARY KEY (name, version)
);
```

Note

Composite primary key is declared as **table constraint**. Each column composing the primary key is said to be a **prime attribute**.

games(<u>name</u>, <u>version</u>, price)

Constraints Check

Recreation

Primary Key

Definition

```
CREATE TABLE games (
  name          VARCHAR(32),
  version       CHAR(3),
  price         NUMERIC,
  PRIMARY KEY (name, version)
);
```

if BOTH are the same, then do not insert

different version

Failure

```
INSERT INTO games VALUES
  ('Aerified', '1.0', 5),
  ('Aerified', '1.0', 6);
```

same

Success

```
INSERT INTO games VALUES
  ('Aerified', '1.0', 5),
  ('Aerified', '2.0', 6),
  ('Verified', '1.0', 7);
```

different name

all different

Constraints Check

Recreation

Not NULL

Code

```
CREATE TABLE games (
  name          VARCHAR(32),
  version       CHAR(3),
  price         NUMERIC NOT NULL,
  PRIMARY KEY (name, version)
);
```

Note

A `NOT NULL` constraint guarantees that no value of the corresponding field in any record of the table can be set to `NULL`. A `NOT NULL` constraint is always declared as a **column constraint**.

When it is explicit, it is declared with the keyword `NOT NULL`. In the example on the right, the price of a game can never be `NULL`.

Constraints Check

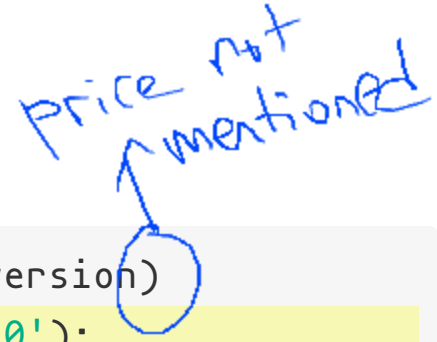
Recreation

Not NULL

Definition

```
CREATE TABLE games (
  name          VARCHAR(32),
  version       CHAR(3),
  price         NUMERIC NOT NULL,
  PRIMARY KEY (name, version)
);
```

Implicit NULL




```
INSERT INTO games (name, version)
  VALUES ('Aerified2', '1.0');
```

Explicit NULL

```
INSERT INTO games VALUES
  ('Aerified2', '1.0', NULL);
```

Constraints Check

Recreation

Not NULL

Default Value

```
CREATE TABLE games (
  name          VARCHAR(32),
  version       CHAR(3),
  price         NUMERIC NOT NULL
    DEFAULT 1.00,
  PRIMARY KEY (name, version)
);
```

Success

```
INSERT INTO games (name, version)
  VALUES ('Aerified2', '1.0');
```

not mentioned, use default

Failure

```
INSERT INTO games VALUES
  ('Aerified2', '1.0', NULL);
```

explicit, not use default

Constraints Check

Recreation

Unique

Code

```
CREATE TABLE customers (
  first_name   VARCHAR(64),
  last_name    VARCHAR(64),
  email        VARCHAR(64) UNIQUE,
  dob          DATE,
  since        DATE,
  customerid   VARCHAR(16),
  country      VARCHAR(16),
  UNIQUE (first_name, last_name)
);
```

Note

A `UNIQUE` constraint on a column or a combination of columns guarantees the table cannot contain two records with the **same value** in the corresponding column or combination of columns.

SQL Standard

According to SQL standard *(and in most system)*, `PRIMARY KEY` is equivalent to `UNIQUE` and `NOT NULL`*.

*There are differences in side-effect: *foreign-key*, *index*, *deferrability*, etc.

Constraints Check

Recreation

Foreign Key

Codes

```
CREATE TABLE downloads (
  customerid  VARCHAR(16)
    REFERENCES customers (customerid),
  name        VARCHAR(32),
  version     CHAR(3),
  FOREIGN KEY (name, version)
    REFERENCES games (name, version)
);
```

Note

Referenced column required to be **primary key** for portability.

```
CREATE TABLE customers (
  first_name  VARCHAR(64),
  last_name   VARCHAR(64),
  email       VARCHAR(64),
  dob         DATE,
  since       DATE,
  customerid  VARCHAR(16) PRIMARY KEY,
  country     VARCHAR(16)
);
CREATE TABLE games (
  name        VARCHAR(32),
  version     CHAR(3),
  price       NUMERIC,
  PRIMARY KEY (name, version)
);
```

Constraints Check

Recreation

Foreign Key

Definition

```
CREATE TABLE downloads (
  customerid  VARCHAR(16)
    REFERENCES customers (customerid),
  name        VARCHAR(32),
  version     CHAR(3),
  FOREIGN KEY (name, version)
    REFERENCES games (name, version)
);
```

Not in Customers

```
INSERT INTO downloads VALUES
  ('Adam1983', 'Aerified', '1.0');
-- ('Adam1983') is not in customers
```

Not in Games

```
INSERT INTO downloads VALUES
  ('Carole89', 'Aerified', '1.1');
-- ('Aerified', '1.1') is not in games
```

Note

We check for the **existence** of the **set of values** in the referenced table.

Constraints Check

Recreation

Foreign Key

Definition

```
CREATE TABLE downloads (
  customerid  VARCHAR(16)
    REFERENCES customers (customerid),
  name        VARCHAR(32),
  version     CHAR(3),
  FOREIGN KEY (name, version)
    REFERENCES games (name, version)
);
```

NULL Customers

```
INSERT INTO downloads VALUES
  (NULL, 'Aerified', '1.0');
-- Allow (NULL) for (customerid)
```

NULL Games

```
INSERT INTO downloads VALUES
  ('Carole89', NULL, '1.1');
-- Allow (NULL, '1.1') for (name, version)
```

Note

We **omit** the check if any value in the set to be checked is NULL.

Constraints Check

Recreation

Foreign Key

Definition

```
CREATE TABLE downloads (
  customerid  VARCHAR(16)
    REFERENCES customers (customerid),
  name        VARCHAR(32),
  version     CHAR(3),
  FOREIGN KEY (name, version)
    REFERENCES games (name, version)
);
```

NULL Customers but Not in Games

```
INSERT INTO downloads VALUES
  (NULL, 'Aerified', '1.1');
-- Check BOTH
```

NULL Games but Not in Customers

```
INSERT INTO downloads VALUES
  ('Adam1983', NULL, '1.1');
-- Check BOTH
```

Note

We check all constraints **independently** and must **satisfy all constraints**.

Constraints Check

Recreation

Foreign Key

Definition

```
CREATE TABLE downloads (
  customerid  VARCHAR(16)
    REFERENCES customers (customerid),
  name        VARCHAR(32),
  version     CHAR(3),
  FOREIGN KEY (name, version)
    REFERENCES games (name, version)
);
```

Prevent Deletion

```
DELETE FROM customers
  WHERE country = 'France';
```

Error

Cannot delete the French customers because some of them have downloaded some games.

Note

The `WHERE <cond>` clause specifies the **condition** for the deletion. In other words, we do not delete all rows, but only rows satisfying the condition `<cond>`.

Constraints Check

Recreation

Check

Positive Price

```
CREATE TABLE games (
  name        VARCHAR(32),
  version     CHAR(3),
  price       NUMERIC NOT NULL
    CHECK (price > 0),
  PRIMARY KEY (name, version)
);
```

Note

A **check** constraint enforces any other condition that can be expressed in SQL.

A check constraint is declared as a **column** or a **table constraint** with the `CHECK` keyword followed by the **SQL condition** in parenthesis. If the condition involves more than one column, it should be a table constraint.

In the example on the right, the price of game can not be zero or negative.

Constraints Check

Recreation

Check

Definition

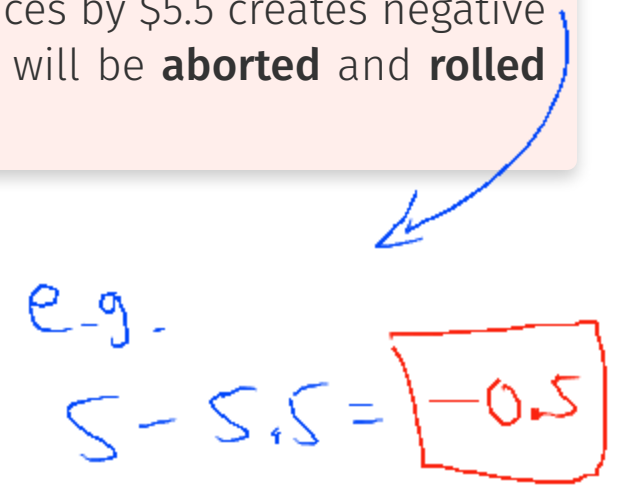
```
CREATE TABLE games (
  name          VARCHAR(32),
  version       CHAR(3),
  price         NUMERIC NOT NULL
    CHECK (price > 0),
  PRIMARY KEY (name, version)
);
```

Prevent Update

```
UPDATE games
  SET price = price - 5.5;
```

Error

Discounting all the prices by $5.5 creates negative prices. This operation will be **aborted** and **rolled back**.



Constraints Check

Recreation

Check

SQL Standard

The **SQL standard** caters for very expression `CHECK` constraints. It even allows that `CHECK` constraints can be implemented as assertions outside tables *(using the construct* ~~`CREATE ASSERTION`~~*)* so that they can **involve several tables** and use **aggregate functions**, **nested queries** *etc.*.

↳ use trigger

In practice, vendors only offer check constraints that are **limited to very simple row and table checks**. PostgreSQL `CHECK` is limited to **boolean expression** involving **one row and one table**.

Vendors usually advocate the use of triggers and stored procedures. But these are notoriously difficult and error-prone. It largely defeats the objective of integrity management.

Constraints Check

Updates

Motivation

Definition

```
CREATE TABLE downloads (
  customerid  VARCHAR(16)
    REFERENCES customers (customerid),
  name        VARCHAR(32),
  version     CHAR(3),
  FOREIGN KEY (name, version)
    REFERENCES games (name, version)
);
```

Question

What happened when things change?

Instances

downloads

name	version	customerid
Skype	1.0	tom1999
Comfort	1.1	john88
Skype	2.0	tom1999

customers

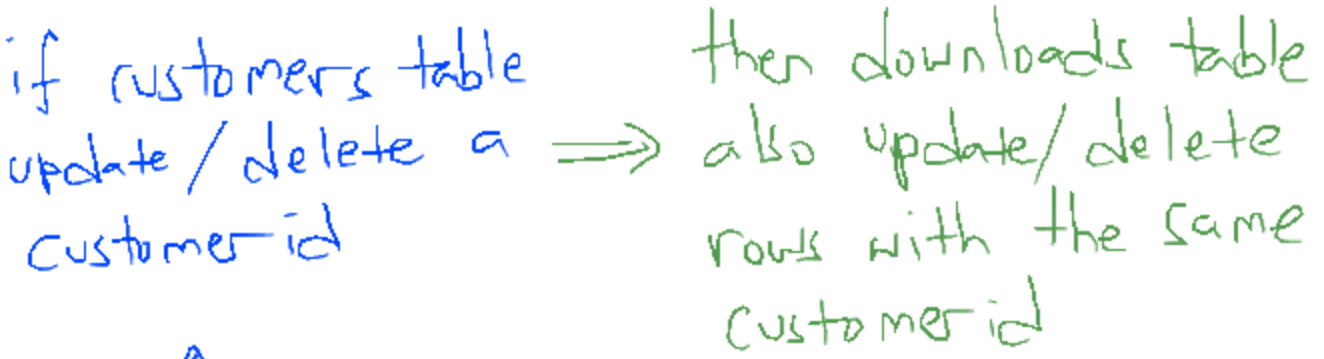
customerid	email	...
tom1999	tlee@gmail.com	...
john88	al@hotmail.com	...
walnuts	dcs@nus.edu.sg	...

Constraints Check

Updates

Action

Cascade




```
CREATE TABLE downloads (
  customerid  VARCHAR(16)
    REFERENCES customers (customerid)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  name        VARCHAR(32),
  version     CHAR(3),
  FOREIGN KEY (name, version)
    REFERENCES games (name, version)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Propagate

Note

`ON UPDATE/DELETE` with the option `CASCADE` propagate the update or deletion.

Other options include:

- `NO ACTION`
- `SET DEFAULT`
- `SET NULL`

May cause **chain reaction**.

*For more powerful generalization, we have **triggers**.

Constraints Check

Advice

Good Practices

Avoid NULL

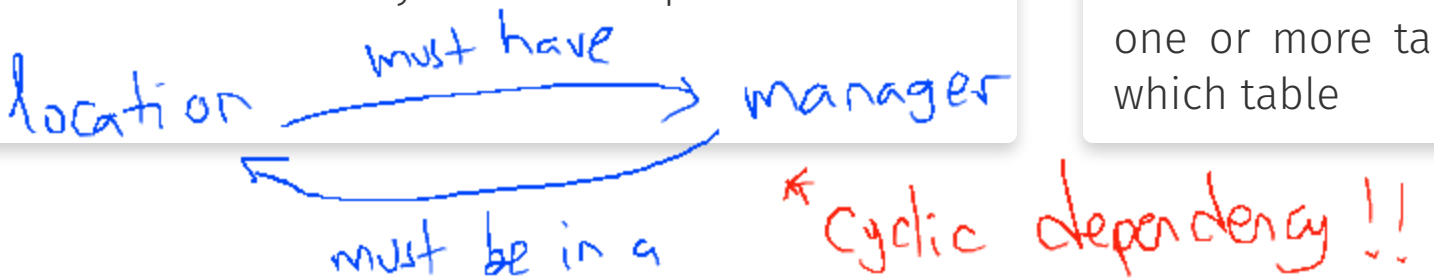
It is generally a good idea to constrain all attributes to be not **NULL** unless there is a good design or tuning reason for not doing so.

Justify Cascade

Think carefully about which foreign keys should be subject to cascade.

Generally Defer

It is generally a good idea to defer all the constraints that can be deferred. A deferred constraint is checked at the end of a transaction and not immediately after each operation.



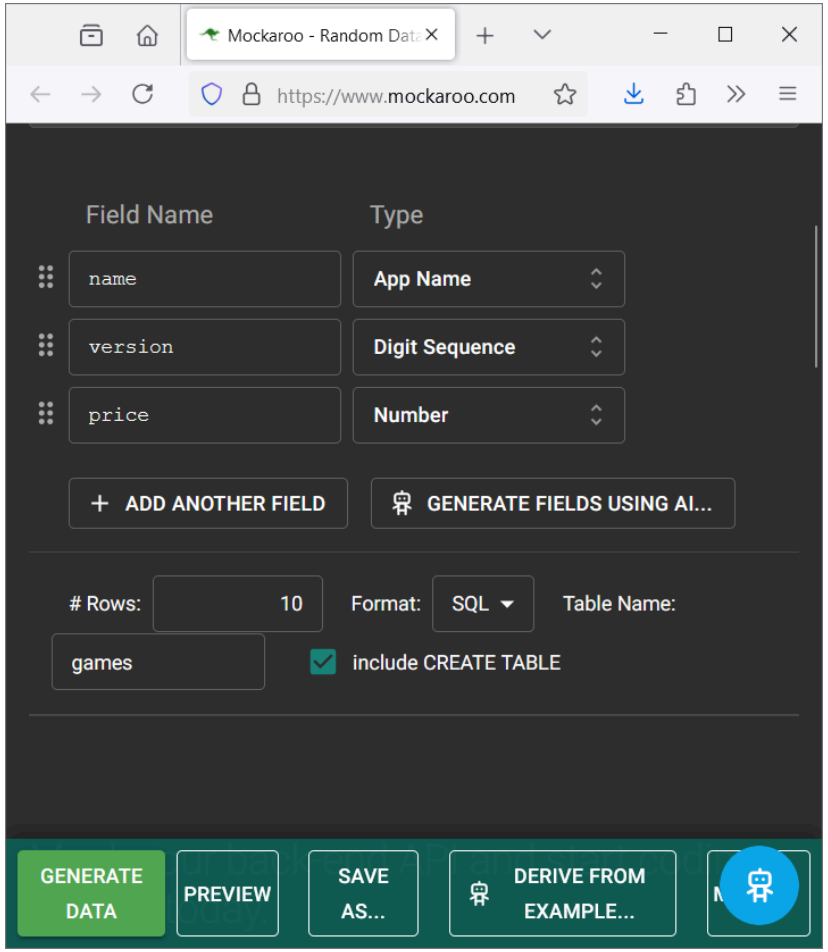
Play the Devil's Advocate

Given the complete schema for the three tables, try and find out the scenario in which an operation *(insertion, deletion, update)* on a table or a transaction containing a set of operation on one or more tables violate which constraint on which table

Query

Mock Data

Mockaroo




```sql
create table games (
  name VARCHAR(50),
  version VARCHAR(50),
  price DECIMAL(3,2)
);
insert into games (name, version, price)
  values ('Sonair', '6.5', 3.68);
insert into games (name, version, price)
  values ('Greenlam', '5.4', 3.91);
insert into games (name, version, price)
  values ('Domainer', '2.9', 8.11);
insert into games (name, version, price)
  values ('Prodder', '6.3', 7.25);
```

App Store

Creating

Steps

1. Download the following files from Canvas "**Files > Cases > AppStore**":
   - `AppStoreSchema.sql`
   - `AppStoreCustomers.sql`
   - `AppStoreGames.sql`
   - `AppStoreDownloads.sql`

2. Open the file and copy the content to **DB Fiddle**.
   - The file `AppStoreClean.sql` is useful to clean up if you run this locally.

Note

We use variants of the same data and that the results on your computer may differ slightly from the results given in the slides for illustration purposes.

Query

App Store

Querying

Prints Customers Table

```
SELECT *
FROM customers;
```

first_name	last_name	email	dob	since	customerid	country
Johnny	Gilbert	jgilberte8@nymag.com	1995-06-19	2023-07-15	JohnnyG89	Malaysia
Deborah	Ruiz	druiz0@drupal.org	1990-08-01	2024-04-17	Deborah84	Singapore
Tammy	Lee	tlee1@barnesandnoble.com	2004-09-14	2024-02-21	Tammy1998	Singapore
…						

1000 rows

Query

App Store

Querying

Prints Games Table

```
SELECT *
FROM games;
```

name	version	price
Aerified	1.0	12
Aerified	1.1	3.99
Aerified	1.2	1.99
…		

430 rows

Query

App Store

Querying

Prints Downloads Table

```
SELECT *
FROM downloads;
```

customerid	name	version
Adam1983	Biodex	1.0
Adam1983	Domainer	2.1
Adam1983	Subin	1.1
...		

4214 rows

```
postgres=# exit
Press any key to continue . . .
```