Database

SQL

Aggregate Queries

Case Study

Game Store

Requirement

Game Store Requirement

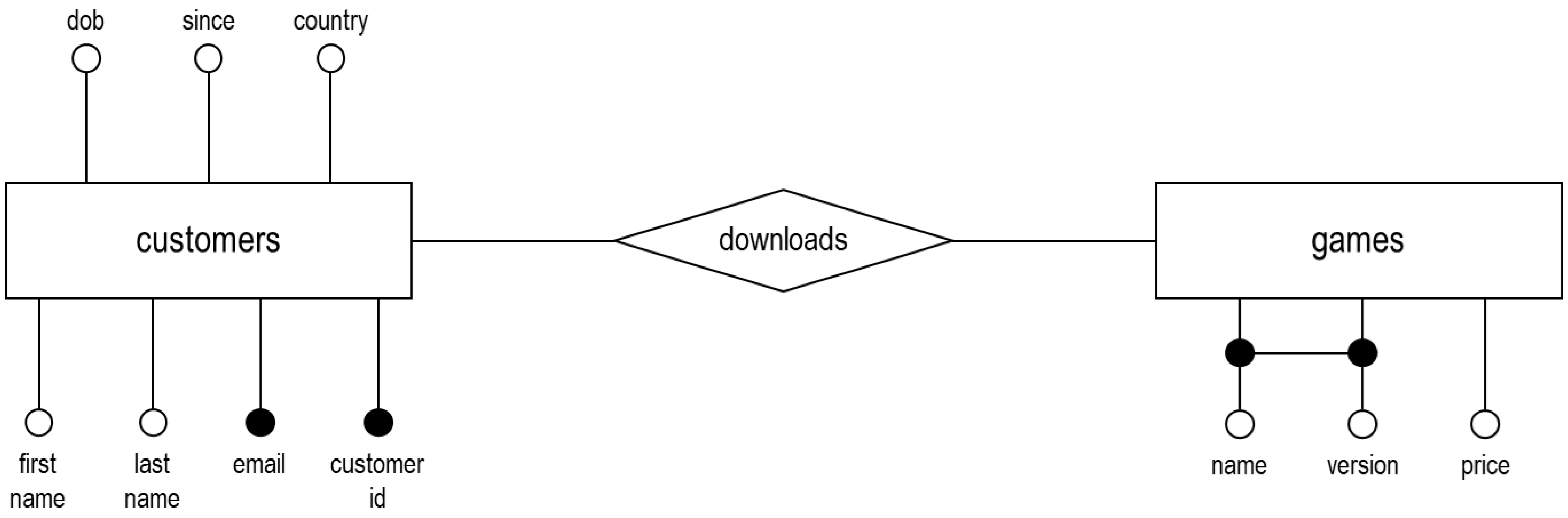
Our company, **Apasaja Pte Ltd**, has been commissioned to develop an application to manage the data of an online app store. We want to store several items of information about our customers such as their first name, last name, date of birth, e-mail, date and country of registration to our online sales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

Case Study

Design

Entity-Relationship Diagram



Aggregation

Functions

Basic

Aggregation Functions

The values of column can be aggregated using **aggregation functions** such as `COUNT()`, `SUM()`, `MAX()`, `MIN()`, `AVG()`, `STDDEV()`, *etc.*. PostgreSQL also allows user-defined aggregate functions.

```
SELECT COUNT(*)
FROM customers c;
```

```
SELECT COUNT(c.customerid)
FROM customers c;
```

count
1000

Note

Count the number of rows in the table.

```
SELECT COUNT(ALL c.country)
FROM customers c;
```

Note

`ALL` is default and often omitted.

Aggregation

Functions

Distinct

DISTINCT Keyword

We need to add the keyword DISTINCT inside the COUNT() aggregate function if we want to count the number of **different** countries in the column country of the table customers.

The keyword DISTINCT can be used in other aggregate functions similarly.

```
SELECT COUNT(DISTINCT c.country)
FROM customers c;
```

count
5

Aggregation

Functions

Example

Aggregate Functions Example

The following query finds the maximum, minimum, average, and standard deviation prices of our games.

It uses the arithmetic TRUNC() to display two decimal places for average and standard deviation.

```
SELECT MAX(g.price), MIN(g.price),
  TRUNC(AVG(g.price), 2) AS avg,
  TRUNC(STDDEV(g.price), 2) AS std
FROM games g;
```

max	min	avg	std
12	1.99	6.97	3.96

Aggregation

Grouping

Logical

GROUP BY

The `GROUP BY` clause creates **logical groups** of records that have the **same values for the specified fields** before computing the aggregate functions.

```
GROUP BY c.country;
```

first_name	last_name	email	...	country
"Deborah"	"Ruiz"	"druiz0@drupal.org"	...	"Singapore"
"Tammy"	"Lee"	"tlee1@barnesandnobles.com"	...	"Singapore"
...				
"Raymon"	"Tan"	"rtan1z@nature.com"	...	"Thailand"
"Jean"	"Ling"	"jlingpn@walmart.com"	...	"Thailand"
...				

Aggregation

Grouping

Aggregation

Aggregation Function Per Group

The aggregation functions are calculated for each logical group.

```sql
SELECT c.country, COUNT(*)
FROM customers c
GROUP BY c.country;
```

country	count
"Vietnam"	98
"Singapore"	391
...	

```sql
SELECT c.country, COUNT(*)
FROM customers c;
/* only one group created */
```

Error

This is actually an error as we cannot select only one value of `c.country`.

Aggregation

Grouping

Where

After WHERE

Groups are formed *(logically)* **after** the rows have been filtered by the `WHERE` clause.

```sql
SELECT c.country, COUNT(*)
FROM customers c
WHERE c.dob >= '2006-01-01'
GROUP BY c.country;
```

country	count
"Vietnam"	4
"Singapore"	25
"Thailand"	5
"Indonesia"	15
"Malaysia"	12

Aggregation

Grouping

From

After FROM

Groups are formed *(logically)* **after** the tables have been joined in the `FROM` clause.

```sql
SELECT c.customerid, c.first_name, c.last_name, SUM(g.price)
FROM customers c, downloads d, games g
WHERE c.customerid = d.customerid
  AND d.name = g.name and d.version = g.version
GROUP BY c.customerid, c.first_name, c.last_name;
```

Note

Find the total spending of each customer.

Aggregation

Grouping

Select

SELECT Clause

It is recommended *(and required per SQL standard)* to **include attributes projected** in the `SELECT` clause by the `GROUP BY` clause.

```
SELECT c.customerid, c.first_name, c.last_name, SUM(g.price)
FROM customers c, downloads d, games g
WHERE c.customerid = d.customerid
  AND d.name = g.name and d.version = g.version
GROUP BY c.customerid;
```

Bad Practice

The above query works only because `first_name` and `last_name` are guaranteed unique.

Aggregation

Grouping

Select

Invalid Query

The following query **does not work** in PostgreSQL *(but works in SQLite with potentially incorrect result)*. We will run all codes on PostgreSQL for testing.

```
SELECT c.customerid, c.first_name, c.last_name, SUM(g.price)
FROM customers c, downloads d, games g
WHERE c.customerid = d.customerid
  AND d.name = g.name and d.version = g.version
GROUP BY c.first_name, c.last_name;
```

Issue

If there are two customers with the same first and last name, which `customerid` is selected?

Aggregation

Grouping

Renaming

Renamed Column

Renamed columns can be used in GROUP BY clause. The following query displays the number of downloads by country and year of birth *(using EXTRACT)*.

```
SELECT c.country, EXTRACT(YEAR FROM c.since) AS regyear, COUNT(*) AS total
FROM customers c, downloads d
WHERE c.customerid = d.customerid
GROUP BY c.country, regyear
ORDER BY regyear ASC, c.country ASC;
```

Grouping

Group Order

GROUP BY Reordering

The order of columns in GROUP BY clause does not change the meaning of the query. The logical groups remain the same.

```sql
SELECT c.country, EXTRACT(YEAR FROM c.since) AS regyear, COUNT(*) AS total
FROM customers c, downloads d
WHERE c.customerid = d.customerid
GROUP BY regyear, c.country
ORDER BY regyear ASC, c.country ASC;
```

Aggregation

Having

Condition

Aggregate Condition

Aggregate functions can be used in **conditions**, but not in WHERE clause. Aggregate functions can be evaluated after groups are formed *(which is after WHERE clause)*.

```
SELECT c.country
FROM customers c
WHERE COUNT(*) >= 100
GROUP BY c.country;
```

HAVING Clause

We need a new clause: HAVING clause. This clause is performed after GROUP BY clause.

HAVING clause can only use aggregate functions, columns listed in the GROUP BY clause, and subqueries.

Aggregation

Having

Condition

Aggregate Condition

Aggregate functions can be used in conditions, but not in WHERE clause. Aggregate functions can be evaluated after groups are formed *(which is after WHERE clause)*.

```
SELECT c.country
FROM customers c
GROUP BY c.country
HAVING COUNT(*) >= 100;
```

Note

The query on the left finds the countries in which there are more than 100 customers.

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