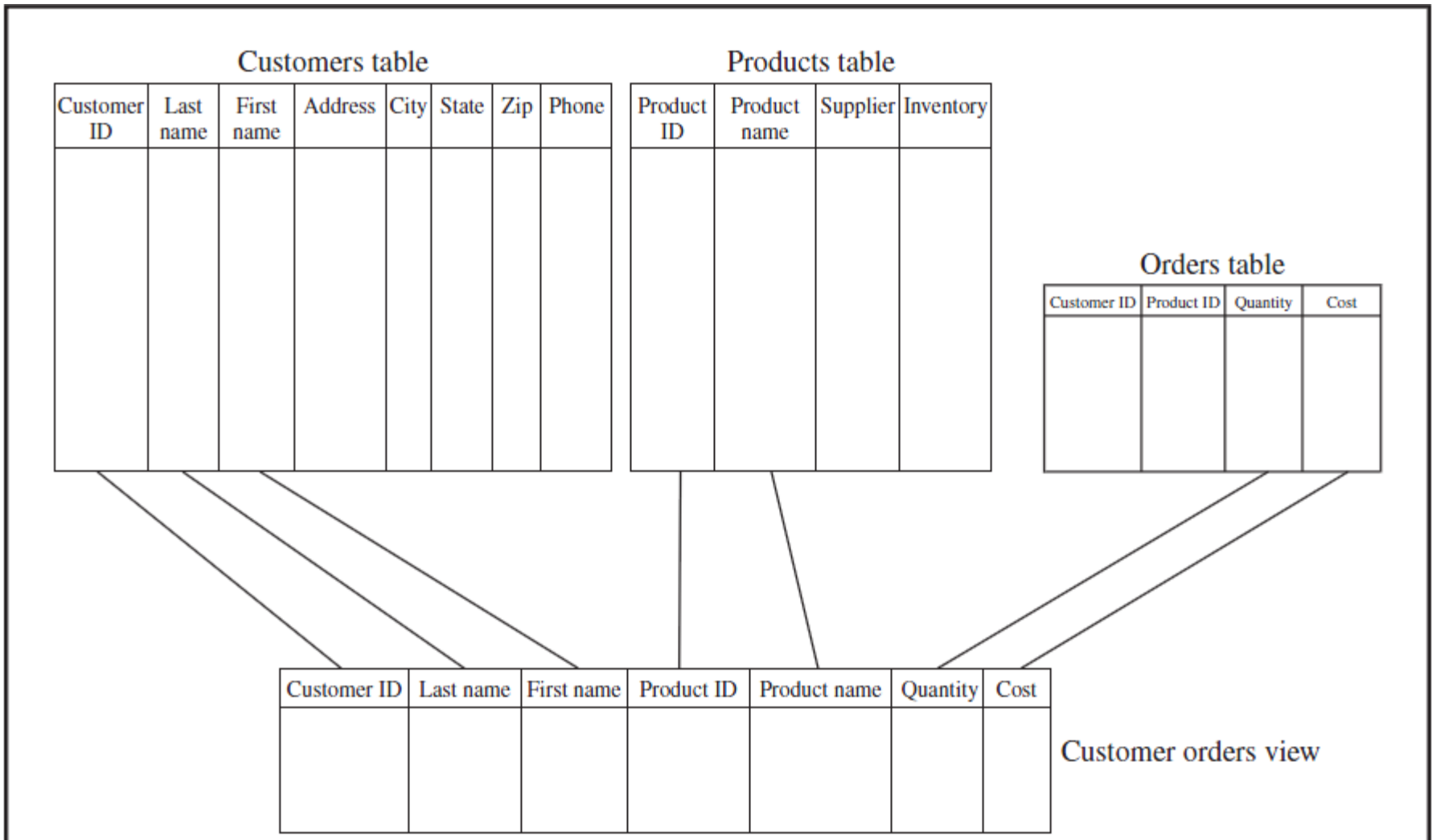
Views



Creating a view from table columns

CREATE VIEW

- CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW *viewname*
  [(*column_name,...])] AS query*
  [WITH [CASCADED | LOCAL] CHECK OPTION]

```
create view store."CustomerOrders" AS
        select "Customer"."LastName", "Customer"."FirstName",
                        "Product"."ProductName", "Order"."TotalCost"
        from store."Order" natural inner join store."Customer"
                        natural inner join store."Product";

grant select on store."CustomerOrders" to "Salesman";
\dv store.
select * from store."CustomerOrders";
```

DROP VIEW

```
DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT
```

`CASCADE`

Automatically drop objects that depend on the view (such as other views), and in turn all objects that depend on those

`RESTRICT`

Refuse to drop the view if any objects depend on it. This is the default.

```
drop view store."CustomerOrders" ;
```

VIEW

- CREATE VIEW defines a view of a query.
- The view is **not physically materialized**. Instead, the query is run every time the view is referenced in a query.

- TEMPORARY or TEMP: Temporary views are **automatically dropped at the end** of the current session

https://www.postgresql.org/docs/13/sql-createview.html

Updatable VIEWs

- A view is automatically updatable (allow INSERT, UPDATE and DELETE statements) if it satisfies all of the following conditions:
  - The view must have exactly one entry in its FROM list, which must be a table or another updatable view.
  - The view definition must not contain WITH, DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clauses at the top level.
  - The view definition must not contain set operations (UNION, INTERSECT or EXCEPT) at the top level.
  - The view's select list must not contain any aggregates, window functions or set-returning functions

Read-only VIEW

- A more complex view, views are **read only:**
  - the system will **not allow an insert, update, or delete on a view**.
  - You can get the effect of **an updatable view** by creating **INSTEAD triggers** on the view, which must ***convert attempted inserts, etc****. on the view into appropriate* ***actions on other tables***

Views - practice

- Use SQL statements to do the following exercises
- Student must save your solution in a sql file:

Studentname_studentID_exercise number.sql

Views - practice

1. Create a view from eduDB, named student_shortinfos, this view contains some information from student table: student_id, firstname, lastname, gender, dob, clazz_id

   1.1. Display all records from this view

   1.2. Try to insert/update/delete a record from student_shortinfos

   - ➔ Check if this record is inserted/updated/deleted from student table

   1.3. Suppose you do not have permission to access student table, but you can access to student_shortinfos view and clazz table, write SQL statements to display

   a) A list of students: student id, fullname, gender and class name.

   b) A list of class (class id, class name) and the number of students in each class.

   1.4. Set address attribute of student table to NOT NULL ➔ then try insert a new record into student_shortinfos view: check if the record is insert into student table

   1.5. Please change dob of a student and check if this infos is also updated in student_shortinfos view.

   1.6. Please insert a new record into student table and then check whether you can see the new student on student_shortinfos view ?

Views - practice

2. Create a view from eduDB, named student_class_shortinfos, this view contains: student_id, firstname, lastname, gender, class name.

   Try to insert/update/delete a record into/from student_class_shortinfos

   - ➔ Check whether this record is inserted/updated/deleted from student table / class table ?

3. Create a view from eduDB, named class_infos, this view contains: class_id, class name, number of students in this class.

   3.1 Display all records from this view.

   3.2 Try to insert/update/delete a record into/from class_infos