Exercises on Relational Databases and SQL queries

This document contains exercises on Relational Databases and SQL queries. Complete the tutorial yourself and answer the question

1. Introduction

A relational database model has data that is represented in tables. We will use Structured Query Language (SQL) as the query language to manipulate and search the data of the database.

The general syntax of the SQL queries looks as follows:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
```

The JOIN operator is used when you want to retrieve data from multiple tables.

Aggregate functions are, for example, “count()”, “avg()”, “sum()”, “min()” and “max()”.

For more information on all operators and functions, visit this website.

2. SQL Fiddle

For the exercises you will use **SQL Fiddle** to create databases online and test SQL queries. It allows you to switch between relational database management systems to run your queries against. Examples of relational database management systems include MySQL, PostgreSQL, Microsoft SQL Server, Microsoft Access, Oracle, and DB2.[1]

3. Entity-Relationship diagrams and schemas

An Entity-Relationship (ER) diagram is a logical representation of a relational database (see **Figure 1**). This logical representation can be used to physically create the database. The resulting diagram is the definition of all table objects, column types, relationships between the tables, functions, constraints, keys and indices. Next, you can create a “schema”, that is a collection of database objects (e.g., tables) that contain the actual/physical data. One database can have multiple database schemas, which in turn contain multiple tables. Note that the same object (e.g., a table named “*table1*”) can be used without conflict by different schemas. Schemas are not rigidly separated: a user can have access to objects in any of the schemas in the database they are connected to as long as they have the privileges to do so. Schemas specify which fields will be present in the database and what will be their data types (e.g., table “*employee*” will have an “*employee_ID*” column represented by a string (varchar) of 10 digits).

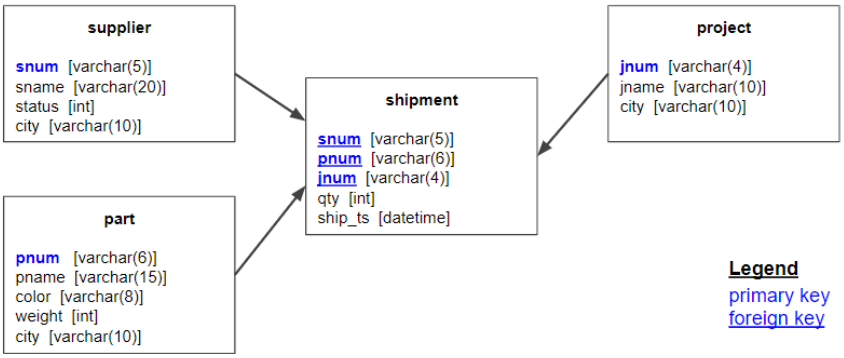


Figure 1: Entity-Relationship diagram of the Supplier Parts Projects (SPJ) database. Entities are "supplier", "part", "shipment" and "project". Primary keys are indicated in bold and foreign keys in bold/underlined.

Definitions:

***Logical model*:** Develop database model based on the requirements. There are no physical implementations (it contains no actual data). For example, creating an Entity-Relationship Model helps to model entities/attributes and relationships.

***Physical model*:** The logical model is implemented and physical tables with real data are created. For example, a "schema" is the physical implementation of a logical model.

3.1 Logical database design: Entity-Relationship diagram

Entity-relationship (ER) diagrams can be used to design relational databases. To illustrate how ER diagrams are created to design relational databases, consider the following case study of "MyFlix", which is a business that rents out movies to people. A database system is created to store and manage movie records. The ER diagram is a logical model of our database and, of course, there is no physical database created yet. An example ER diagram for "MyFlix" is shown in **Figure 2**.

Entities are real-world things and can be conceptual (e.g., sales orders). In this example, the entities to be included are:

- Members with member information
- Movies with movie information
- Categories with information on movie categories
- Movie rentals with info on rented movies to members
- Payments with payment information

Each entity has attributes (represented by columns in each table) with a specific data type. For example, the entity "members" has the following attributes:

- Membership number "*membership_number*" (Primary key, INT)
- Full names "*name*" (VARCHAR(45))
- Gender "*gender*" (bit(1))
- Date of birth "*birth*" (Date)
- Physical address "*physic_ad*" (VARCHAR(45))
- Postal address "*post_ad*" (VARCHAR(45)))

It is important to choose the right data type for each of the attributes. Also, make sure to select the correct primary key for each entity.

Relationship information (1:1, 1:n or n:m relationships) between the entities:

- A member can rent more than one movie in the same period
- A movie can be rented by more than one member in a given period
- A movie can only belong to one category
- Categories can have more than one movie
- A member can only have one account
- A member can make a number of payments

Which type of relationship (1:1, 1:n or n:m) is most appropriate between “Members” and “Movie rentals”? And between “Movie rentals” and “Movies”?

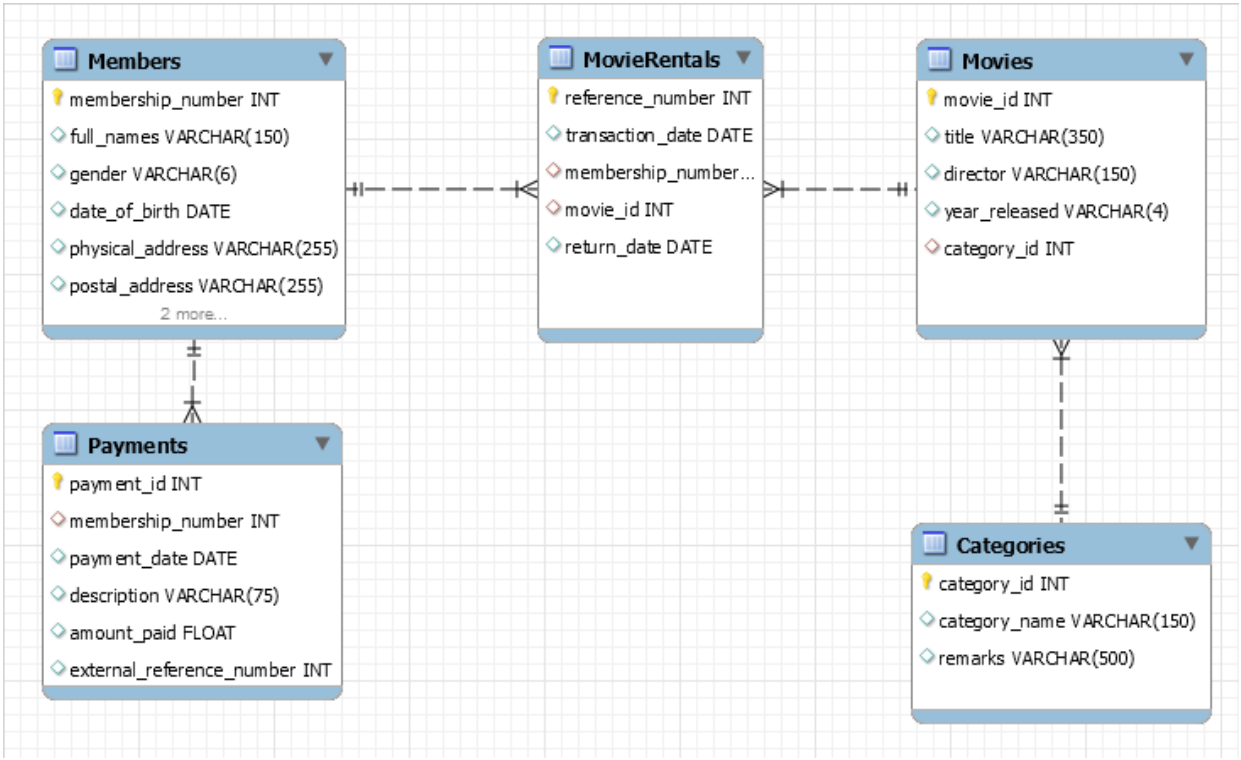


Figure 2: Entity-Relationship diagram for “MyFlix” database.

3.2 Schemas

First, go to http://sqlfiddle.com/. To create a schema, you can execute SQL scripts in the left panel in SQL Fiddle (the “schema” panel).

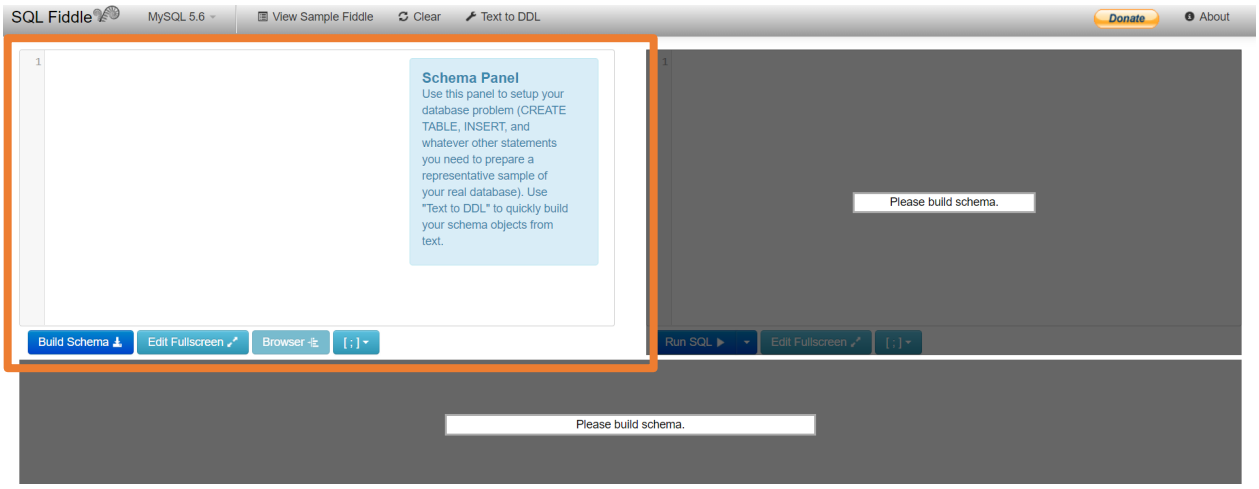


Figure 3: SQL Fiddle. On the left, the "schema" panel (to create schemas and insert data). On the right, the "SQL query" panel to run SQL queries. Below the "output" panel to see results from queries.

3.3 Create schemas and insert data

We create a database called "Supplier Parts Projects" (SPJ), which is a simple example database made available by Date (2004) to illustrate relational databases. The database contains four tables:

- Table "**supplier**": information about suppliers. The "*snum*" identifies the supplier, while the other attributes each hold one piece of information about the supplier.
- Table "**part**": information about the parts. The "*pnum*" identifies the parts, while the other attributes each hold one piece of information about the part.
- Table "**project**": information about the projects. The "*jnum*" identifies the projects, while the other attributes each hold one piece of information about the project for which the part is used.
- Table "**shipment**": information about the shipments. The "*snum*", "*pnum*" and "*jnum*" identify each shipment, while the remaining attribute indicates how many parts were shipped. It is assumed that only one shipment exists for each supplier/part/project pairing (which is not very realistic in real-world scenarios).

To create the database and insert the data, paste the commands in **spj_database.sql** into the left panel of SQL Fiddle. The following tables will be created:

SUPPLIER

SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

PART

PNUM	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

PROJECT

JNUM	JNAME	CITY
J1	Sorter	Paris
J2	Display	Rome
J3	OCR	Athens
J4	Console	Athens
J5	RAID	London
J6	EDS	Oslo
J7	Tape	London

SHIPMENT

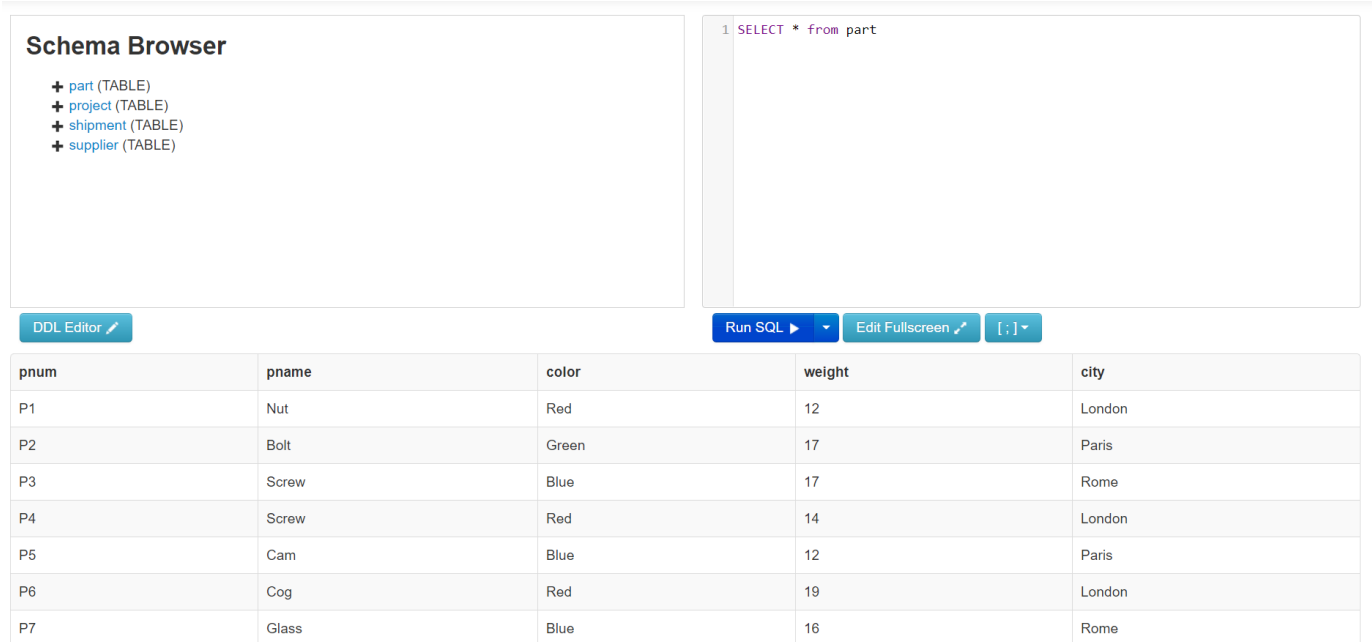
SNUM	PNUM	JNUM	QTY	SHIP_TS
S1	P1	J1	200	2020-02-24 11:10
S1	P1	J4	700	2020-02-24 13:30
S2	P3	J1	400	2020-02-24 12:15
S2	P3	J2	200	2020-02-24 14:40
S2	P3	J3	200	2020-02-25 11:55
S2	P3	J4	500	2020-02-25 15:00
S2	P3	J5	600	2020-02-25 15:20
S2	P3	J6	400	2020-02-26 16:30
S2	P3	J7	800	2020-02-26 16:45
S2	P5	J2	100	2020-02-26 17:50
S3	P3	J1	200	2020-02-24 11:35
S3	P4	J2	500	2020-02-24 14:15
S4	P6	J3	300	2020-02-24 14:05
S4	P6	J7	300	2020-02-25 12:55
S5	P2	J2	200	2020-02-24 11:45
S5	P2	J4	100	2020-02-24 14:40
S5	P5	J5	500	2020-02-24 16:50
S5	P5	J7	100	2020-02-25 12:05
S5	P6	J2	200	2020-02-25 13:00
S5	P1	J4	100	2020-02-25 15:50
S5	P3	J4	200	2020-02-25 17:25
S5	P4	J4	800	2020-02-26 11:55
S5	P5	J4	400	2020-02-26 14:40
S5	P6	J4	500	2020-02-26 15:25

Figure 4: SPJ_DB.xlsx screenshot.

What are the primary keys of each of the created tables? What are the foreign keys? Make sure you know the difference between primary and foreign keys.

To query the database, you can use the right panel on SQL Fiddle (the “SQL query” panel). The panel below is the “output” panel where you see results from your queries.

An example query to select all data from Table “*part*” is shown in **Figure 6**.



pnum	pname	color	weight	city
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London
P7	Glass	Blue	16	Rome

Figure 6: Example query in SQL Fiddle to return the “*part*” Table.

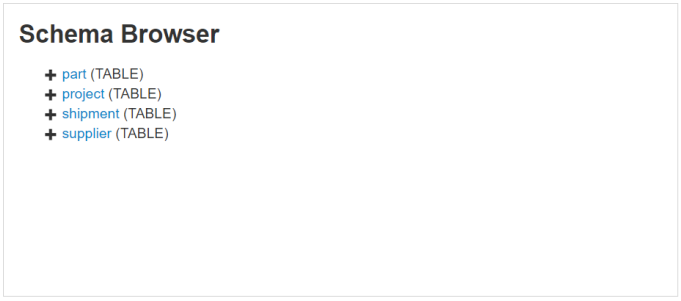
4. Exercises

A) SPJ database

Use the SPJ database for this exercise. The database can be created using the script **spj_database.sql**. Paste the commands into SQL Fiddle.

Example queries:

Example 1) *Show the name of parts in a first column and the total shipped quantity per part in a second column. Sort the table by increasing shipped quantity. Change the column names respectively to 'part name' and 'total shipped quantity'.*

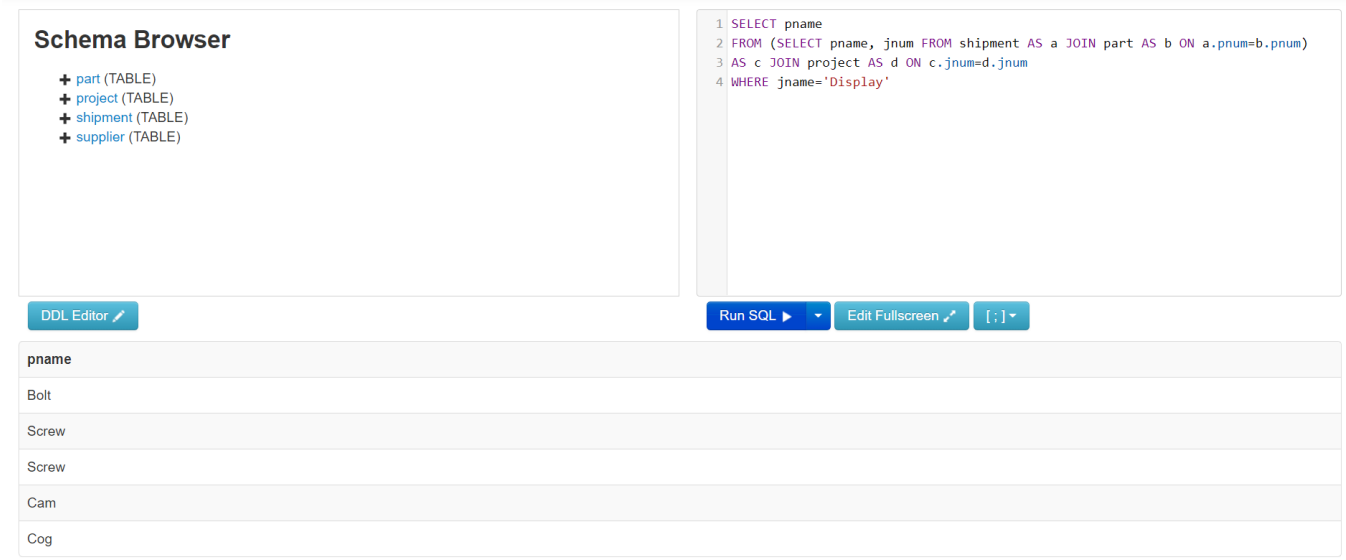



```
select pname as 'part name', sum(qty) as 'total shipped quantity'
from part as a
join shipment as b on a.pnum = b.pnum
group by a.pnum
order by sum(qty) asc
```

DDL Editor | Run SQL | Edit Fullscreen | [;]

part name	total shipped quantity
Bolt	300
Nut	1000
Cam	1100
Cog	1300
Screw	1300
Screw	3500

Example 2) *Show all parts (names) used in project Display.*



pname
Bolt
Screw
Screw
Cam
Cog

Exercises:

Exercise A.1: Make the necessary adjustments or create commands to insert two extra rows in the Table "*part*":

P7, Glass, Blue, 16, Rome

P8, Sensor, Yellow, 14, Paris

Exercise A.2: Select the names and corresponding cities of all suppliers.

Exercise A.3: Select shipments where the shipped quantity is at least 500.

Exercise A.4: Show all names of the projects that are based in London.

Exercise A.5: Give all shipments for project J7 where the quantity is at least 200.

Exercise A.6: Select parts with a name that starts with the letter "C", with a weight of at least 15 and with a city that is not Paris. (**Hint:** use a *like "C*"* where "C*" is a regular expression that matches any string starting with C)

Exercise A.7: Show the total weight of parts per city.

Exercise A.8: Select suppliers based in London and sort the data by increasing value of status. (**Hint:** use ORDER BY operator.)

Exercise A.9: Show the total quantities per shipment where you exclude the shipments with part "P1". Show only the shipments with a total quantity larger than 1000. (**Hint:** the WHERE operator cannot be used for aggregate functions).

Exercise A.10: Show all shipments for project Tape with a minimum part quantity of 200. (**Hint:** use a JOIN operator.)

Exercise A.11: Show all part names shipped from London.

Exercise A.12: Show for each shipment the total shipped weight for projects in Paris.

Exercise A.13: Show the total weight of parts for each project. The first column is the project name, the second column is the total weight of parts.

Exercise A.14: Show all pairs of parts that are within the same city. (**Hint:** use a JOIN operator.) Do not show pairs of a part with itself.

B) MyFlix

For this exercise, use the "MyFlix" database. This database can be created with the script **MyFlix_database.sql**.

Exercises:

Exercise B.1: Select the names, gender and physical address of all members.

Exercise B.2: Return a list of all members showing the membership ID, full names and year of birth. (**Hint**: use LEFT() function).

Exercise B.3: Return a list of movies from the database where you show the movie title and year of release in one field. Show the year in brackets. In a second column, show the director of that movie. (**Hint:** look at "concat()" function.)

Exercise B.4: Show a list of members of the MyFlix database who are referred to "MyFlix" movie rental company and show in a second column the name of the person that referred them. Change column names respectively to "member" and "referred_by" using aliases.

Exercise B.5: Show a list of names of MyFlix members with movies they rented and the category of the movie.

Advanced queries:

This exercise consists of harder queries, combining joins, group by and nested queries and **are typical exam questions.**

Exercise B.6: Give the names of all directors who have created movies **from all** categories

Exercise B.7: Give the titles of all movies who been rented at least 3 times.

Exercise B.8: Give all members who rented at least one movie direct by Rob Marshall but never rented any movie from Nicholas Stoller.

Solutions advances queries:

Solution B.6: Using count distinct in a nest

```
select director from movies m1 where
(select count(distinct(category_id)) as cnt
 from movies m2
 where m2.director = m1.director) =  (select count(category_id)  from categories)
```

Solution B.6: Using not exists and not in

```
select director from movies m1
where not exists
(select category_id from categories
 where category_id not in
  (select category_id from movies m2
   where m2.director = m1.director))
```

Solution B.7: Using a **view** to make things simpler

```
create view movie_rental_counts as
 (select movie_id, count(reference_number) as no_of_rentals
 from movierentals
 group by movie_id);
select title
from movies, movie_rental_counts
where movies.movie_id = movie_rental_counts.movie_id
and no_of_rentals > 2
```

Solution B.7: Using **with** statement

```
with movie_rental_counts as
 (select movie_id, count(reference_number) as no_of_rentals
 from movierentals
 group by movie_id)
```

```
select title
from movies, movie_rental_counts
where movies.movie_id = movie_rental_counts.movie_id and no_of_rentals > 2
```

Solution B.8:

```
select * from members m
where
  exists (select movie_id
        from movierentals natural join movies
        where director = "Rob Marshall"
        and movierentals.membership_number = m.membership_number)
  and not exists
       (select movie_id
        from movierentals natural join movies
        where director = "Nicholas Stoller"
        and movierentals.membership_number = m.membership_number)
```