MINISTRY OF EDUCATION AND SCIENCE OF THE REPUBLIC OF KAZAKHSTAN

International University of Information Technology

Faculty of Information Technology

Department of Information Systems

**Project**«Amazing Events»

**Data and information management**

***Students:*** Yenglik Zhubanova (27325), Azhar Kurmangalieva (27734),  
Assel Temirkhan (27439)

***Teacher:*** Madina B. Aitbekova

Almaty, 2022

**Content**

[**Description of project 3**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938142)

[Amazing Events 3](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938144)

[Amazing events Management System 3](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938146)

[**Logical design 3**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938143)

[Entities and attributes 3](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938144)

[Weak and strong entities 3](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938146)

[Four types of attributes 4](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938146)

**Relationships: PK, FK** [**5**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938147)

[**Normalization 6**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938153)

[**Specialization and Generalization 7**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938152)

[**ER-Diagram 8**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938154)

[**SQL – Create Tables 9**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938142)

[**SQL – Foreign Key 10**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938143)

**All tables are created**  [**11**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938147)

[Database diagram 12](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938149)

[Add Contraints 12](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938150)

[**Insert Statements**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938152) **13**

[**Select 17**](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938154)

[Update 19](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938149)

[Delete 19](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938150)

[View 20](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938150)

DBMS and Driver [22](file:///C:\Users\Assel\Downloads\Project_part1.docx#_Toc97938150)

**Description of project**

***Amazing events***

«Amazing events» is large company that organizes events in New York. During the whole year, the company holds a variety of events. These include fashion shows, meetings and interviews with celebrities, musical extravaganzas, exhibitions, fairs and charity performances.

***Amazing events Management System***

«Amazing events» organizes events of various types. Information about the various types of events is stored in the file named **Event Types**. This file contains information including codes of events and description of the activity of the corresponding type. Every client who wants to organize an event should provide information about the features of this event. Along with the transfer of information it makes the payment for the event. Payment is paid partly, according to the price list of fees. The client must pay all of the payment before or on the day of the event.

All information related to the **event** - the event id, the id of the event type, the event location, status, start date, end date, description of event, number of participants - are stored in the file named Events.

**Payment** information, including the amount of the payment, discount, the date of payment, method of payment code and a description of the method of payment, is also stored in the Events file.

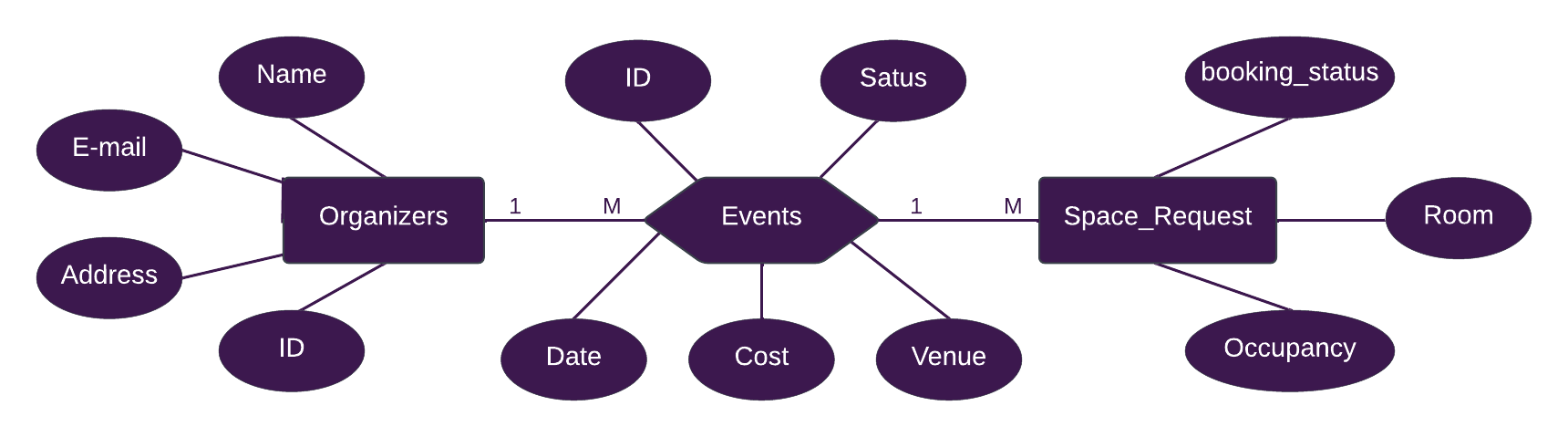
The **client** information - client id, client name and address are in the file named Client.

Each event is managed by a certain **organizer** of «Amazing events». Organizer information - organizer id, name, e-mail is stored in the file named Organizers

**Logical design**

***Entities and attributes***

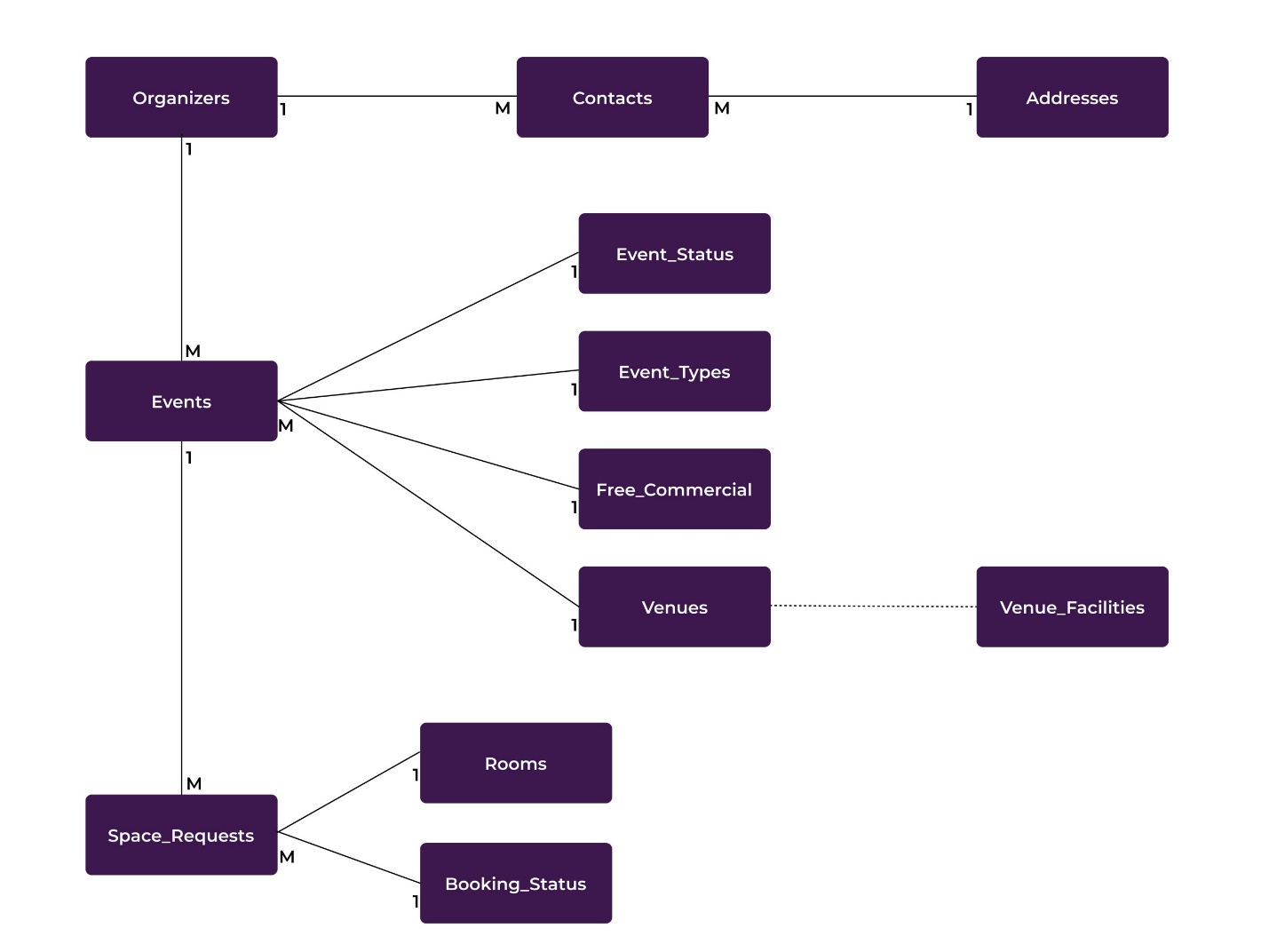
In the ER diagram for our project, the main entity is **Events**, and the rest in the form of ***Organizers, Space\_Request*** are dependent and auxiliary entities. Each of these entities has its own attributes.



***Weak and strong entities***

Any object in the system that we want to model and store information in it is an entity. We have 2 types of entities: weak and strong.

In our information system, the **Venues** is a strong entity, and the **Venue\_facilities** is a weak entity. Because this entity may or may not exist. And the rest are strong subjects.



***There are*** ***four types of attributes****:*

**Simple attributes** are separate components of atomic and elemental. For example, in our system there are objects "Event\_Status" and "Booking\_Status" with attributes: "Event\_Status\_Description", "booking\_status\_description".

**Composite attributes** consist of multiple components, each component being simple or complex. In our case, the table "Addresses" consists of two attributes: "City" and "Street".

**Multi-valued attributes** have multiple values for a particular object. Attribute addresses are created using multi-valued attributes and compound attributes and are called compound attributes. In the tables "Contacts", "Addresses" and "Organizers" e-mail ddress and telephone may use more than one value.

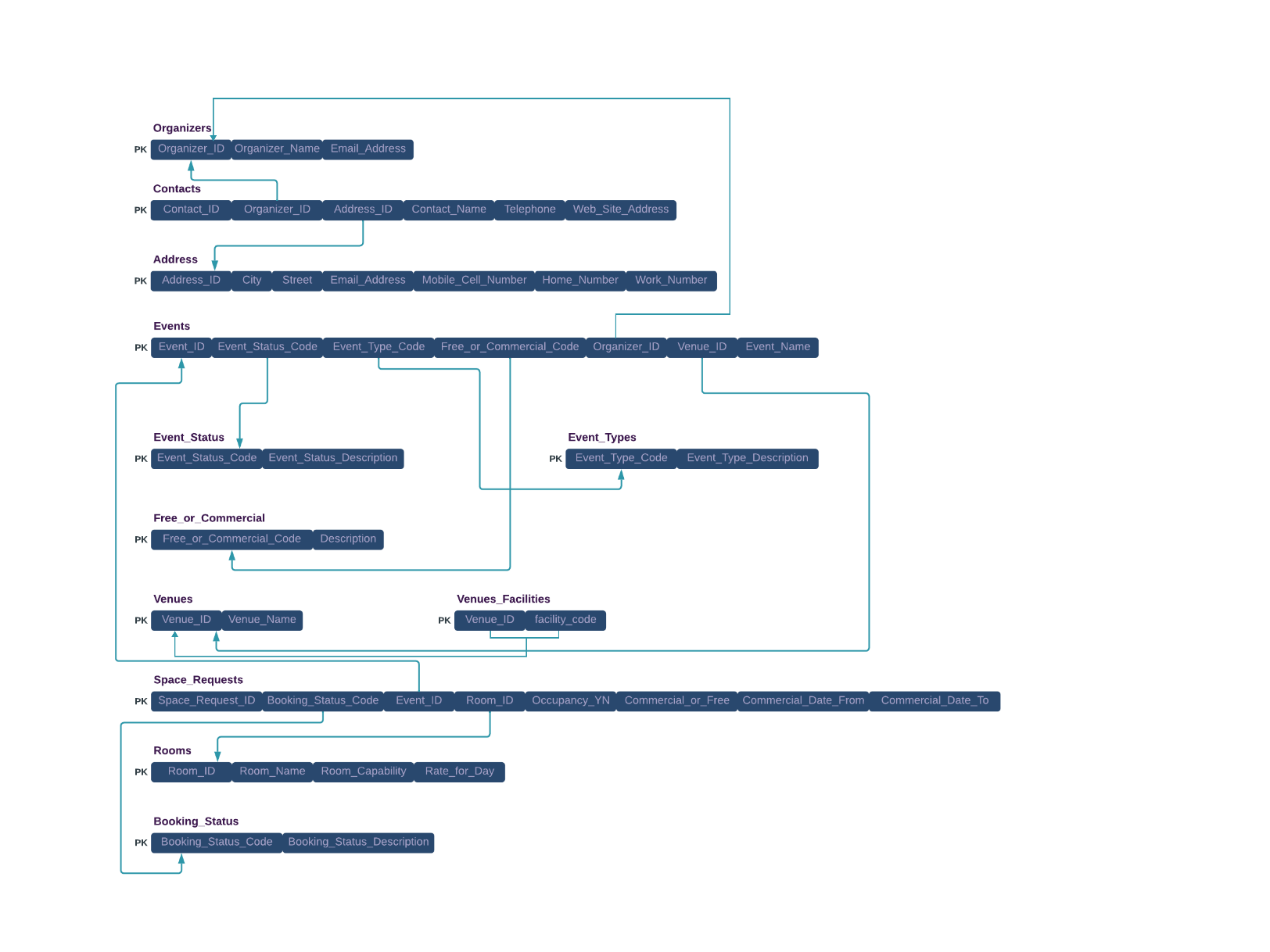
**Derived attributes** are attributing whose values are calculated from other attributes. If an attribute's value can be determined from the values of other attributes, then the attribute is derivable. In the "Client" object the attribute "Age" can be counted from the current date by using "Birth\_date". Hence, the "Age" attribute is derived one.

**Relationships: PK, FK**

|  |  |
| --- | --- |
| **Table name** | **Foreign key** |
| *Contacts* | *Organizer\_ID* |
| *Address\_ID* |
| *Events* | *Organizer\_ID* |
| *Event\_Status\_Code* |
| *Event\_Type\_Code* |
| *Free\_or\_Commercial\_Code* |
| *Venue\_ID* |
| *Space\_Requests* | *Event\_ID* |
| *Booking\_status\_code* |
| *Room\_ID* |

In our database, we tried to avoid many-to-many relationships and stick to the one-to-many structure.

|  |  |
| --- | --- |
| **Table name** | **Primary key** |
| *Organizers* | *Organizer\_ID* |
| *Addresses* | *Address\_ID* |
| *Events* | *Event\_ID* |
| *Event\_Status* | *Event\_Status\_Code* |
| *Event\_Type* | *Event\_Type\_Code* |
| *Free\_or\_Commercial* | *Free\_or\_Commercial\_Code* |
| *Venues* | *Venue\_ID* |
| *Rooms* | *Room\_ID* |
| *Booking\_Status* | *Booking\_status\_code* |
| *Space\_Requests* | *Space\_Request\_ID* |
| *Contacts* | *Contact\_ID* |



**Normalization**

**Third normal form** (**3NF**) is a [normal form](https://en.wikipedia.org/wiki/Database_normalization#Normal_forms) that is used in [normalizing](https://en.wikipedia.org/wiki/Database_normalisation) a [database](https://en.wikipedia.org/wiki/Database) design to reduce the duplication of data and ensure [referential integrity](https://en.wikipedia.org/wiki/Referential_integrity) by ensuring that:-

* the entity is in [second normal form](https://en.wikipedia.org/wiki/Second_normal_form)
* all the attributes in a table are determined only by the [candidate keys](https://en.wikipedia.org/wiki/Candidate_key) of that relation and not by any non-prime attributes.

3NF was designed to improve database processing while minimizing storage costs. 3NF data modeling was ideal for [online transaction processing (OLTP)](https://en.wikipedia.org/wiki/Online_transaction_processing) applications with heavy order entry type of needs.



Functional dependency:

FD1: **{Organizer\_id}**->{Organizer\_Name,Email-Address}

FD2: **{Contact\_ID}**->{ Organizer\_id ,Address\_ID,Contact\_Name,Telephone,Web\_Site\_Address,}

FD3: **{Address\_ID}**->{City,Street,Email-Address,Mobile\_Cell\_Number,Home\_Number,Work\_Number }

FD4**:{Event\_ID}**->{Event\_Status\_Code,Event\_Type\_Code,Free\_or\_Commercial\_Code,Organizer\_ID,Venue\_ID,Event\_Name,Event\_Status\_Date,Event\_End\_Date,Number\_of\_Participants,Derrived\_Days\_Duration, Discount, Amount\_Paid, Cost, Comments}

FD5: **{Event\_Satus\_Code}**->{ Event\_Satus\_Description}

FD6: **{Event\_Type\_Code}**->{ Event\_Type\_Description}

FD7: **{Free\_or\_Commercial\_Code}**->{Description}

FD8: **{Venue\_ID}**->{ Venue\_Name}

FD9: **{Venue\_ID,facility\_code}**->{}

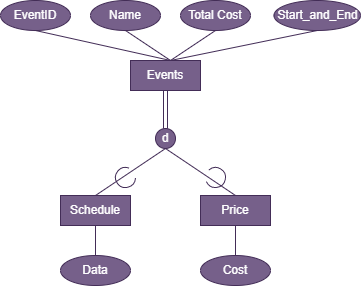
FD10: **{Space\_Request \_ID}**->{Booking\_Status\_Code,Event\_ID,Room\_ID,Occupancy\_YN,Commercial\_or\_Free,Commercial\_Date\_From,Commercial\_Date\_To}

FD11: **{ Booking\_Status\_Code }**->{ Booking\_Status\_Description}

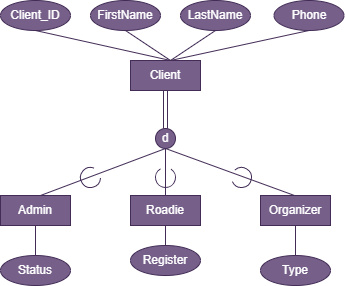
FD12: **{Room\_ID}**->{ Room\_Name,Room\_Capability,Rate\_for\_Day}

**Specialization and Generalization**

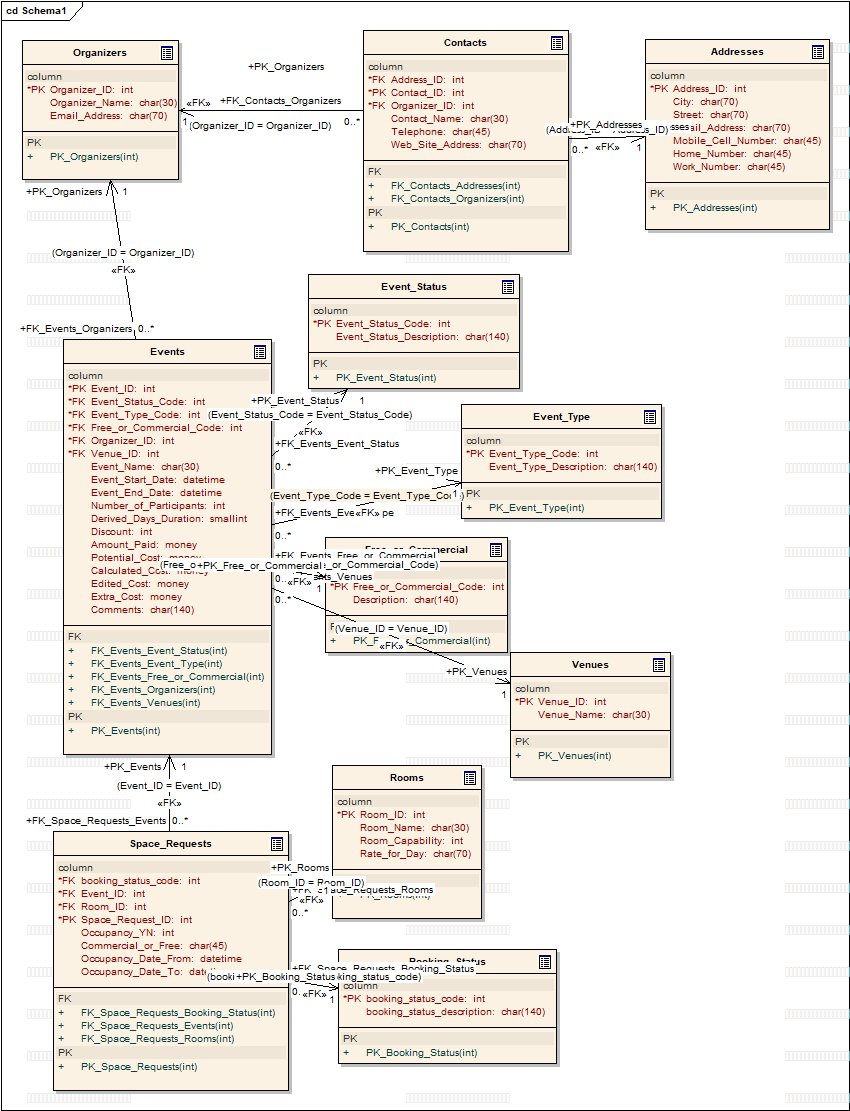
**Generalization** is the process of extracting shared characteristics from two or more classes, and combining them into a **generalized** superclass. Shared characteristics can be attributes, associations, or methods.



**Specialization** means creating new subclasses from an existing class.



**ER-Diagram**



**SQL – Create Tables**

CREATE TABLE Addresses (

Address\_ID int NOT NULL,

City char(70),

Street char(70),

Email\_Address char(70),

Mobile\_Cell\_Number char(45),

Home\_Number char(45),

Work\_Number char(45));

CREATE TABLE Booking\_Status (

booking\_status\_code int NOT NULL,

booking\_status\_description char(140));

CREATE TABLE Contacts (

Address\_ID int NOT NULL,

Contact\_ID int NOT NULL,

Organizer\_ID int NOT NULL,

Contact\_Name char(30),

Telephone char(45),

Web\_Site\_Address char(70));

CREATE TABLE Event\_Status (

Event\_Status\_Code int NOT NULL,

Event\_Status\_Description char(140));

CREATE TABLE Event\_Type (

Event\_Type\_Code int NOT NULL,

Event\_Type\_Description char(140));

CREATE TABLE Events (

Event\_ID int NOT NULL,

Event\_Status\_Code int NOT NULL,

Event\_Type\_Code int NOT NULL,

Free\_or\_Commercial\_Code int NOT NULL,

Organizer\_ID int NOT NULL,

Venue\_ID int NOT NULL,

Event\_Name char(30),

Event\_Start\_Date datetime,

Event\_End\_Date datetime,

Number\_of\_Participants int,

Derived\_Days\_Duration smallint,

Discount int,

Amount\_Paid money,

Potential\_Cost money,

Calculated\_Cost money,

Edited\_Cost money,

Extra\_Cost money,

Comments char(140));

CREATE TABLE Free\_or\_Commercial (

Free\_or\_Commercial\_Code int NOT NULL,

Description char(140));

CREATE TABLE Organizers (

Organizer\_ID int NOT NULL,

Organizer\_Name char(30),

Email\_Address char(70));

CREATE TABLE Rooms (

Room\_ID int NOT NULL,

Room\_Name char(30),

Room\_Capability int,

Rate\_for\_Day char(70));

CREATE TABLE Space\_Requests (

booking\_status\_code int NOT NULL,

Event\_ID int NOT NULL,

Room\_ID int NOT NULL,

Space\_Request\_ID int NOT NULL,

Occupancy\_YN int,

Commercial\_or\_Free char(45),

Occupancy\_Date\_From datetime,

Occupancy\_Date\_To datetime);

CREATE TABLE Venues (

Venue\_ID int NOT NULL,

Venue\_Name char(30));

**SQL – Foreign Key**

ALTER TABLE Addresses ADD CONSTRAINT PK\_Addresses

PRIMARY KEY (Address\_ID);

ALTER TABLE Booking\_Status ADD CONSTRAINT PK\_Booking\_Status

PRIMARY KEY (booking\_status\_code);

ALTER TABLE Contacts ADD CONSTRAINT PK\_Contacts

PRIMARY KEY (Contact\_ID);

ALTER TABLE Event\_Status ADD CONSTRAINT PK\_Event\_Status

PRIMARY KEY (Event\_Status\_Code);

ALTER TABLE Event\_Type ADD CONSTRAINT PK\_Event\_Type

PRIMARY KEY (Event\_Type\_Code);

ALTER TABLE Events ADD CONSTRAINT PK\_Events

PRIMARY KEY (Event\_ID);

ALTER TABLE Free\_or\_Commercial ADD CONSTRAINT PK\_Free\_or\_Commercial

PRIMARY KEY (Free\_or\_Commercial\_Code);

ALTER TABLE Organizers ADD CONSTRAINT PK\_Organizers

PRIMARY KEY (Organizer\_ID);

ALTER TABLE Rooms ADD CONSTRAINT PK\_Rooms

PRIMARY KEY (Room\_ID);

ALTER TABLE Space\_Requests ADD CONSTRAINT PK\_Space\_Requests

PRIMARY KEY (Space\_Request\_ID);

ALTER TABLE Venues ADD CONSTRAINT PK\_Venues

PRIMARY KEY (Venue\_ID);

ALTER TABLE Contacts ADD CONSTRAINT FK\_Contacts\_Addresses

FOREIGN KEY (Address\_ID) REFERENCES Addresses (Address\_ID);

ALTER TABLE Contacts ADD CONSTRAINT FK\_Contacts\_Organizers

FOREIGN KEY (Organizer\_ID) REFERENCES Organizers (Organizer\_ID);

ALTER TABLE Events ADD CONSTRAINT FK\_Events\_Event\_Status

FOREIGN KEY (Event\_Status\_Code) REFERENCES Event\_Status (Event\_Status\_Code);

ALTER TABLE Events ADD CONSTRAINT FK\_Events\_Event\_Type

FOREIGN KEY (Event\_Type\_Code) REFERENCES Event\_Type (Event\_Type\_Code);

ALTER TABLE Events ADD CONSTRAINT FK\_Events\_Free\_or\_Commercial

FOREIGN KEY (Free\_or\_Commercial\_Code) REFERENCES Free\_or\_Commercial (Free\_or\_Commercial\_Code);

ALTER TABLE Events ADD CONSTRAINT FK\_Events\_Organizers

FOREIGN KEY (Organizer\_ID) REFERENCES Organizers (Organizer\_ID);

ALTER TABLE Events ADD CONSTRAINT FK\_Events\_Venues

FOREIGN KEY (Venue\_ID) REFERENCES Venues (Venue\_ID);

ALTER TABLE Space\_Requests ADD CONSTRAINT FK\_Space\_Requests\_Booking\_Status

FOREIGN KEY (booking\_status\_code) REFERENCES Booking\_Status (booking\_status\_code);

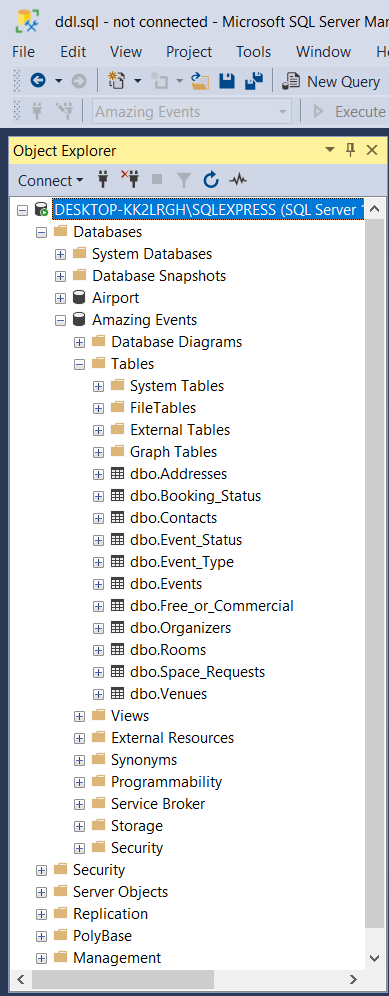
ALTER TABLE Space\_Requests ADD CONSTRAINT FK\_Space\_Requests\_Events

FOREIGN KEY (Event\_ID) REFERENCES Events (Event\_ID);

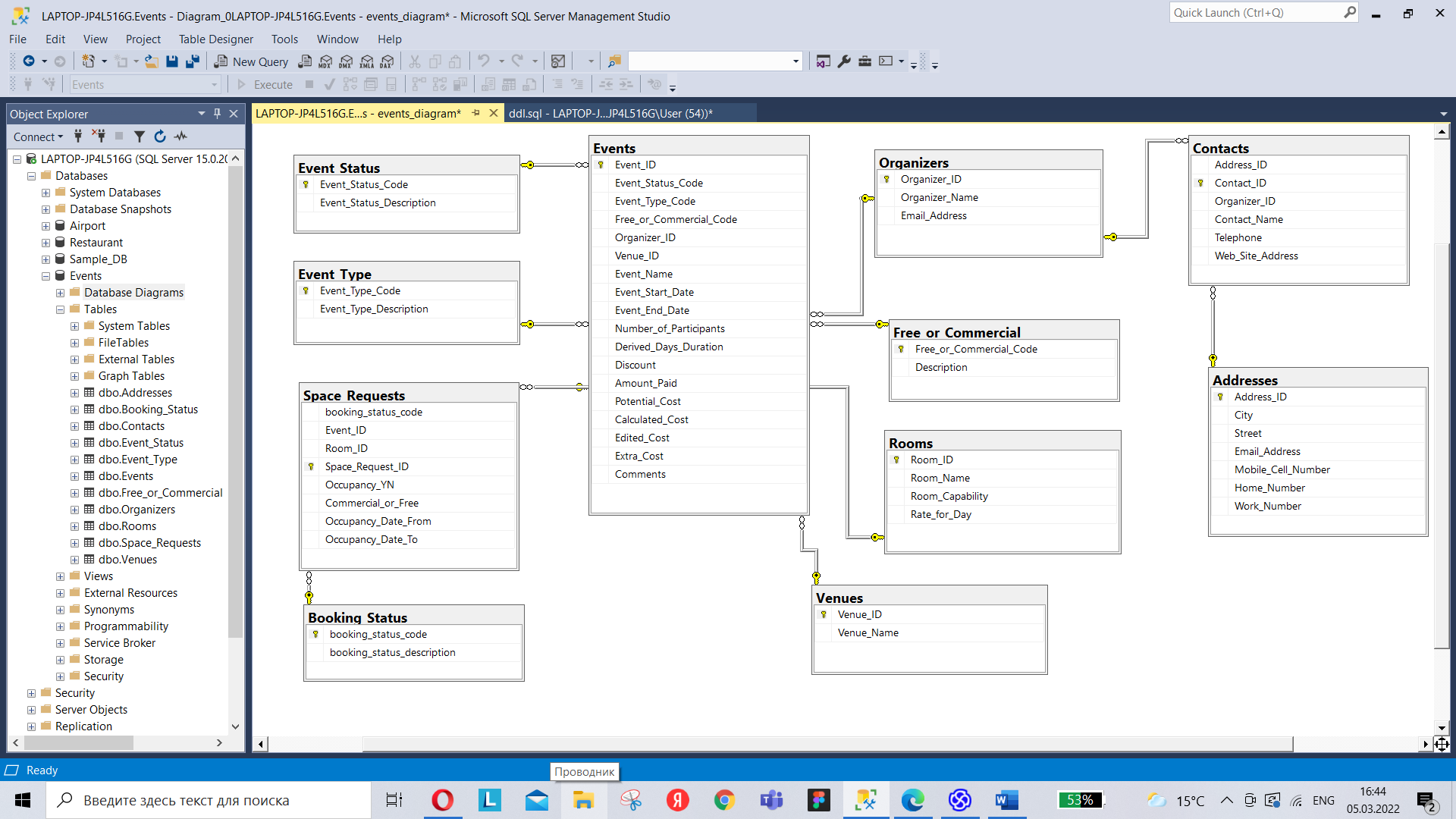
ALTER TABLE Space\_Requests ADD CONSTRAINT FK\_Space\_Requests\_Rooms

FOREIGN KEY (Room\_ID) REFERENCES Rooms (Room\_ID);

**All tables are created**



**Database diagram**



**Add Constraints**

Constraints in SQL are rules and constraints that are set on one or more tables. Such restrictions are necessary to maintain the integrity, accuracy, and reliability of the data in the table. In other words, if the entered values match the restriction rule, they will be successfully entered into the table itself. Otherwise, if the entered values do not match the conditions in the constraint rule, the data will not be entered into the table and the operation will be aborted. There are 6 main limitations in SQL Server:

1. SQL NOT NULL

This restriction prevents a NULL value from being inserted into a table column. When populating data into a table, this constraint indicates an error if the corresponding column has not been populated.

CREATE TABLE Event\_Status (

Event\_Status\_Code int NOT NULL,

Event\_Status\_Description char(140));

1. UNIQUE

This constraint is used to prevent duplicate values by making the data in the table unique. In other words, the UNIQUE constraint ensures that no rows have the same value.

ALTER TABLE Contacts ADD CONSTRAINT UQ\_Contacts\_Contact\_ID

UNIQUE(Contact\_ID);

1. PRIMARY KEY

This constraint is used to identify each row in the table. Due to the PRIMARY KEY constraint, a table with this value cannot be NULL and duplicate values are not allowed when populating data in the table.

ALTER TABLE Events ADD CONSTRAINT PK\_Events

PRIMARY KEY (Event\_ID);

1. FOREIGN KEY

A FOREIGN KEY constraint is used to identify values in a table for a relationship between other tables by referencing a column in child columns that contains a foreign key. This can be seen in our example below.

ALTER TABLE Contacts ADD CONSTRAINT FK\_Contacts\_Addresses

FOREIGN KEY (Address\_ID) REFERENCES Addresses (Address\_ID);

1. CHECK

The CHECK constraint is designed to limit the range of values when entering data into a table by using a certain condition.

*Thanks to the "LIKE" operator, we specify that when entering data into the table, check the mail for the presence of the "@" sign, otherwise the data entry will not be allowed and an error will be displayed.*

ALTER TABLE Addresses ADD CONSTRAINT CK\_Email\_Address

CHECK(Email\_Address LIKE '%@%');

1. DEFAULT

This constraint helps to populate the table with the default values that were specified in the condition. It will fill a row in the table with a default value if no value’s specified when you enter it.

*In our case, thanks to the DEFAULT constraint, we specify that if the string for the date is not filled, it will default to the current date using the GETDATE() statement.*

ALTER TABLE Events ADD CONSTRAINT CK\_Events\_StarDate

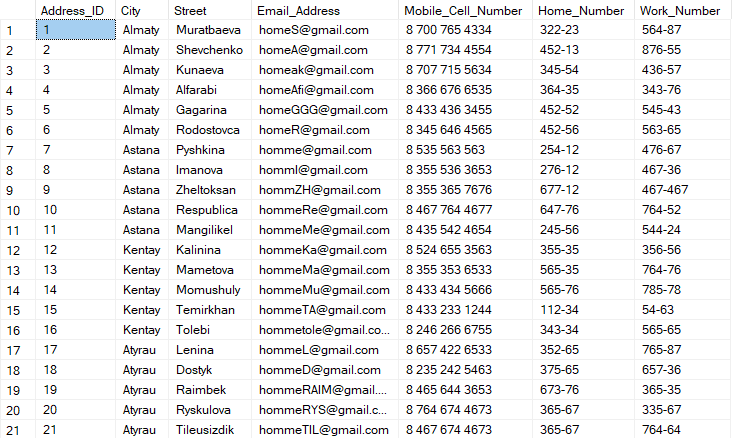
DEFAULT GETDATE() FOR Event\_Start\_Date;

ALTER TABLE Events ADD CONSTRAINT CK\_Events\_EndDate

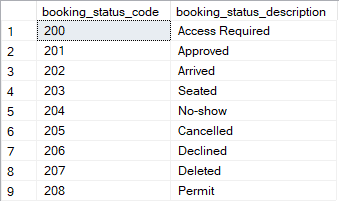
DEFAULT GETDATE() FOR Event\_End\_Date;

**Insert Statements**

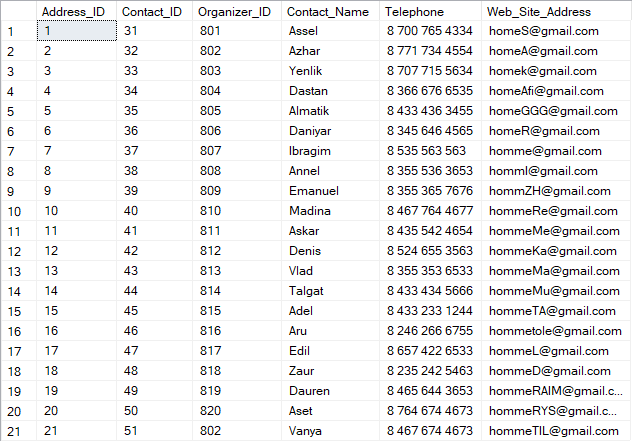
* **Addresses**



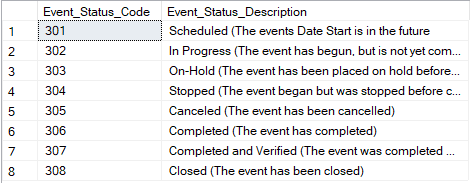
* **Booking\_Status**



* **Contacts**



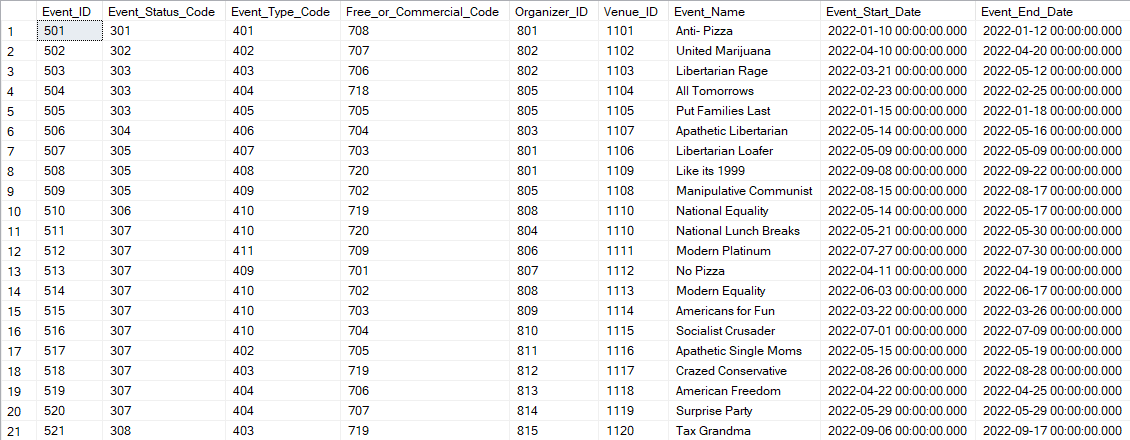
* **Event\_Status**

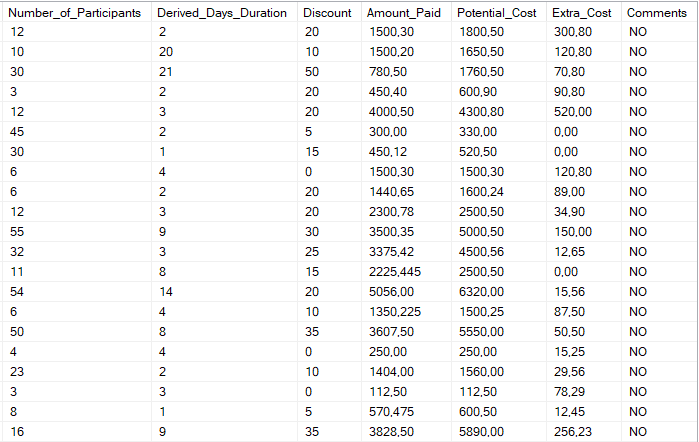


* **Event\_Type**

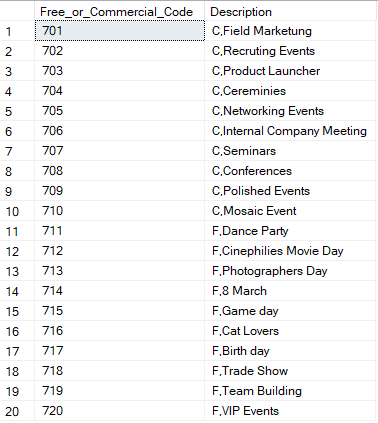


* **Events**





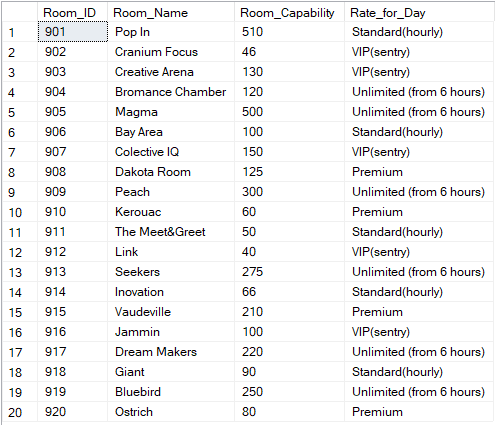
* **Free\_or\_Commercial**



* **Organizers**



* **Rooms**



* **Space\_Requests**



* **Venues**



**Select**

--1. Display the first and last date of the event for each tariff with their count and sum of amount where the payment for the event exceeds 700 dollars

SELECT R.Rate\_for\_Day,

count(\*) as count,

min(SR.Occupancy\_Date\_From) as first\_date\_from,

max(SR.Occupancy\_Date\_To) as end\_date\_to,

sum(E.Amount\_Paid) as sum\_amount

FROM Rooms R JOIN Space\_Requests SR ON R.Room\_ID=SR.Room\_ID

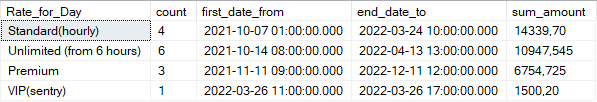
JOIN Events E ON SR.Event\_ID=E.Event\_ID

where E.Amount\_Paid>700.0

GROUP BY R.Rate\_for\_Day

ORDER BY first\_date\_from

Result:



(4 rows affected)

Completion time: 2022-04-18T22:51:45.8016989+06:00

--2. Find past events in ascending order when the occupancy of a place request was less than 500

SELECT DISTINCT SR.Event\_ID,E.Event\_Name,E.Event\_Start\_Date,E.Event\_End\_Date,E.Number\_of\_Participants,E.Derived\_Days\_Duration,E.Amount\_Paid,SR.Commercial\_or\_Free FROM Events E,Space\_Requests SR WHERE

SR.Event\_ID IN(SELECT Event\_ID FROM Space\_Requests

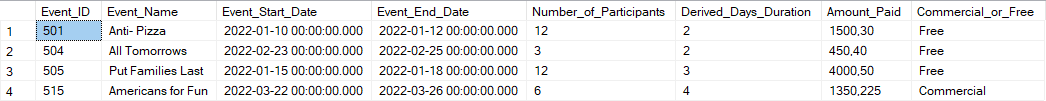
GROUP BY Event\_ID HAVING COUNT(Space\_Requests.Occupancy\_YN)<500) AND

E.Event\_End\_Date<'2022-04-19 00:00:00' AND

E.Event\_ID=SR.Event\_ID

ORDER BY Event\_ID;

Result:



(4 rows affected)

Completion time: 2022-04-18T22:55:58.0610303+06:00

--3. Print the name of the client indicating his city and organizer, where the potential cost exceeds the extra costs, sorted in descending order of the potential cost

SELECT A.City,C.Contact\_Name,O.Organizer\_Name, E.Potential\_Cost FROM Addresses A

JOIN Contacts C ON A.Address\_ID=C.Address\_ID

JOIN Organizers O ON C.Organizer\_ID=O.Organizer\_ID

JOIN Events E ON O.Organizer\_ID=E.Organizer\_ID

WHERE Potential\_Cost>(SELECT SUM(Extra\_Cost)FROM Events)

ORDER BY Potential\_Cost DESC

Result:



(8 rows affected)

Completion time: 2022-04-19T03:00:33.0062656+06:00

--4. Display in one table the average amount paid by booking code and individual values of the amount paid of events grouped by booking code where booking status starts with D

SELECT Event\_Name,BS.booking\_status\_code,Amount\_Paid FROM Events E

JOIN Space\_Requests SR ON E.Event\_ID=SR.Event\_ID

JOIN Booking\_Status BS ON SR.booking\_status\_code=BS.booking\_status\_code

WHERE booking\_status\_description LIKE 'D%'

UNION

(SELECT 'Z-AVG' AS Event\_Name,BS.booking\_status\_code,AVG(Amount\_Paid) AS Amount\_Paid FROM Events E

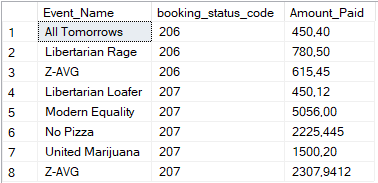
JOIN Space\_Requests SR ON E.Event\_ID=SR.Event\_ID

JOIN Booking\_Status BS ON SR.booking\_status\_code=BS.booking\_status\_code

WHERE booking\_status\_description LIKE 'D%'

GROUP BY BS.booking\_status\_code)

ORDER BY BS.booking\_status\_code,Event\_Name



(8 rows affected)

Completion time: 2022-04-19T03:26:53.8132501+06:00

**Update**

--1. Increase by 30% the potential cost and amount paid for events where the name contains American

UPDATE Events

SET Potential\_Cost=1.3\*Potential\_Cost,Amount\_Paid=1.3\*Amount\_Paid

WHERE Event\_Name LIKE '%American%'

(2 rows affected)

Completion time: 2022-04-19T02:09:31.8764057+06:00

--2. Change the occupancy for an event with ID 520 or with a occupancy date from January 20 at 12 p.m.

UPDATE Space\_Requests

SET Occupancy\_YN=130

WHERE Event\_ID=520

OR Occupancy\_Date\_From = '2022-01-20 12:00:00'

(2 rows affected)

Completion time: 2022-04-19T02:17:46.5909811+06:00

--3. Update the contact name from id 44 to Asset and number to 8 701 469 58 89

UPDATE Contacts

SET Contact\_Name='Asset', Telephone='8 701 469 58 89'

WHERE Contact\_ID=44

(1 row affected)

Completion time: 2022-04-19T02:30:37.8725291+06:00

--4. Update from address mobile number to contact telephones for all addresses with an ID less than 7

UPDATE Addresses

SET Mobile\_Cell\_Number = C.Telephone

FROM Addresses A

INNER JOIN Contacts C

ON (A.Email\_Address = C.Web\_Site\_Address)

WHERE A.Address\_ID <7 ;

(5 rows affected)

Completion time: 2022-04-19T02:44:16.6547981+06:00

**Delete**

--1. Remove a description from a free or commercial table that has a Day

DELETE FROM Free\_or\_Commercial

WHERE Description LIKE '%Day%'

(4 rows affected)

Completion time: 2022-04-19T03:37:13.6977822+06:00

--2. Delete customers whoms name's leght is 5 letters and starts from A

DELETE FROM Contacts

WHERE len(Contact\_Name) = 5

AND Contact\_Name LIKE 'A%'

(5 rows affected)

Completion time: 2022-04-19T13:15:48.4493579+06:00

--3. Remove from Venues where Venue name include word Park

DELETE FROM Venues

WHERE Venue\_Name LIKE '%Park'

(4 rows affected)

Completion time: 2022-04-19T13:21:47.9592773+06:00

--4. Remove Events which price is low than average cost of full events

DELETE FROM Events

WHERE Amount\_Paid < (SELECT AVG(Amount\_Paid) FROM Events)

(13 rows affected)

Completion time: 2022-04-19T13:20:11.3105898+06:00

**View**

--1. Event, start and end date of the event, number of participants more than 20, event status, venue name, discount on the event

SELECT \* FROM Eventdetails

CREATE VIEW Eventdetails

AS

SELECT dbo.Events.Event\_Name,

dbo.Events.Event\_Start\_Date,

dbo.Events.Event\_End\_Date,

dbo.Events.Number\_of\_Participants,

dbo.Event\_Status.Event\_Status\_Description,

dbo.Venues.Venue\_Name,

((dbo.Events.Potential\_Cost\*dbo.Events.Discount)/100) as Discount\_Cost

FROM dbo.Event\_Status INNER JOIN dbo.Events ON

dbo.Event\_Status.Event\_Status\_Code = dbo.Events.Event\_Status\_Code INNER JOIN

dbo.Venues ON dbo.Events.Venue\_ID = dbo.Venues.Venue\_ID

WHERE dbo.Events.Number\_of\_Participants > 20

GROUP BY dbo.Events.Event\_Name,

dbo.Events.Event\_Start\_Date,

dbo.Events.Event\_End\_Date,

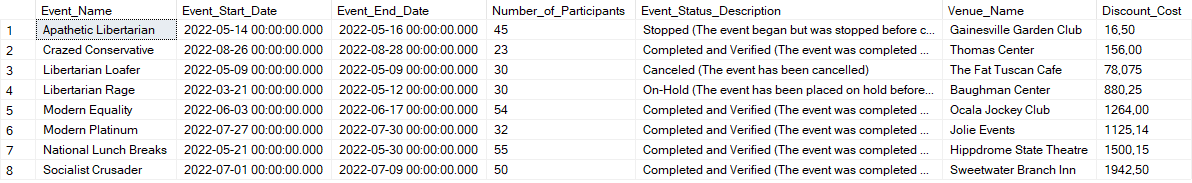
dbo.Events.Number\_of\_Participants,

dbo.Event\_Status.Event\_Status\_Description,

dbo.Venues.Venue\_Name,

((dbo.Events.Potential\_Cost\*dbo.Events.Discount)/100)

Result:



(8 rows affected)

Completion time: 2022-04-18T23:30:30.9748400+06:00

--2. Booking status and its count, average occupancy and sum number of participants for each status, where the duration of the event days is more than a week

SELECT \* FROM Booking

CREATE VIEW Booking

AS

SELECT BS.booking\_status\_description,count(\*) AS count, AVG(SR.Occupancy\_YN) AS avg\_occupancy,SUM(E.Number\_of\_Participants) as sum\_num\_of\_participant

FROM Booking\_Status BS

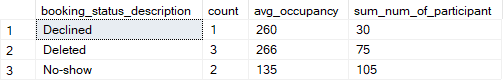
INNER JOIN Space\_Requests SR ON BS.booking\_status\_code = SR.booking\_status\_code

INNER JOIN Events E ON SR.Event\_ID = E.Event\_ID

WHERE E.Derived\_Days\_Duration >7

GROUP BY BS.booking\_status\_description

Result:



(3 rows affected)

Completion time: 2022-04-19T00:05:20.1423422+06:00

--3. Declined booking status where Organizer is Jon Morris

SELECT \* FROM DeclinedStatus\_details

CREATE VIEW DeclinedStatus\_details

AS

SELECT Contact\_Name, Telephone, City,booking\_status\_description, Organizers.Organizer\_ID, Organizer\_Name

FROM Contacts

JOIN Addresses ON Contacts.Address\_ID = Addresses.Address\_ID

JOIN Booking\_Status ON Booking\_Status.booking\_status\_description = Booking\_Status.booking\_status\_description

JOIN Organizers ON Contacts.Organizer\_ID = Organizers.Organizer\_ID

WHERE booking\_status\_description = 'Declined' AND Organizers.Organizer\_ID = 802

Result:



(1 row affected)

Completion time: 2022-04-19T13:25:13.5055922+06:00

--4. Event's price is above the average price, and has status Completed

SELECT \* FROM EventsAboveaveragePrice

CREATE VIEW EventsAboveaveragePrice

AS

SELECT Event\_Name, Amount\_Paid, Event\_Status\_Description,Event\_Type\_Description

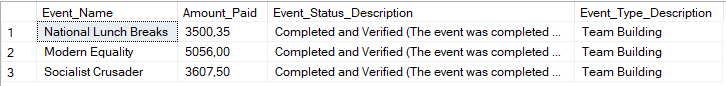
FROM Events

JOIN Event\_Type ON Events.Event\_Type\_Code = Event\_Type.Event\_Type\_Code

JOIN Event\_Status ON Events.Event\_Status\_Code = Event\_Status.Event\_Status\_Code

WHERE Amount\_Paid > (SELECT AVG(Amount\_Paid) FROM Events) AND Event\_Status\_Description LIKE '%Completed%'

Result:



(3 rows affected)

Completion time: 2022-04-19T13:32:26.0526301+06:00

--5. Find amont of duration where cost is more than 1550

SELECT \* FROM AmountOfDuration

CREATE VIEW AmountOfDuration

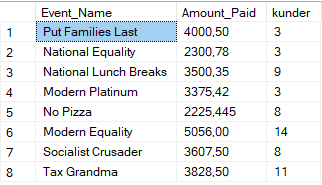
AS

SELECT Event\_Name, Amount\_Paid, DATEDIFF(DAY, Event\_Start\_Date, Event\_End\_Date) as kunder

FROM Events

WHERE Amount\_Paid > 1550

Result:



(8 rows affected)

Completion time: 2022-04-19T13:35:18.5756928+06:00

**DBMS and Driver**

**Microsoft SQL Server** is a relational database management system developed by Microsoft Corporation. It allows you to connect to it locally or over the network, send a command using a special TDS protocol and, accordingly, receive a response. This server supports remote connections, works with many popular data types, makes it possible to create triggers and stored data, has practical and convenient utilities for configuration;

SQL Server is characterized by such features as:

* Efficiency. SQL Server is very fast.
* Reliability and safety. SQL Server provides data encryption.
* Simplicity. This DBMS is relatively easy to work with and administer.

**JDBC – Java DataBase Connectivity** – connection to Java databases. This is the standard of interaction with the database. Its interface comes in a standard JDK in the form of a java.sql package. Whatever library you use to work with the database, it is almost always based on low-level JDBC.

JDBC driver is a JDBC implementation for a specific database. Several different drivers can be registered in the application. When connecting to the database, the desired one is selected based on the connection URL.