CS4D2A SQL Project

Exchanging Contact Details at Events by Wearing a Wristband and Shaking Hands

Sean Rowan – 12100404 – 4th Year Electronic and Computer Engineering – 7/12/15

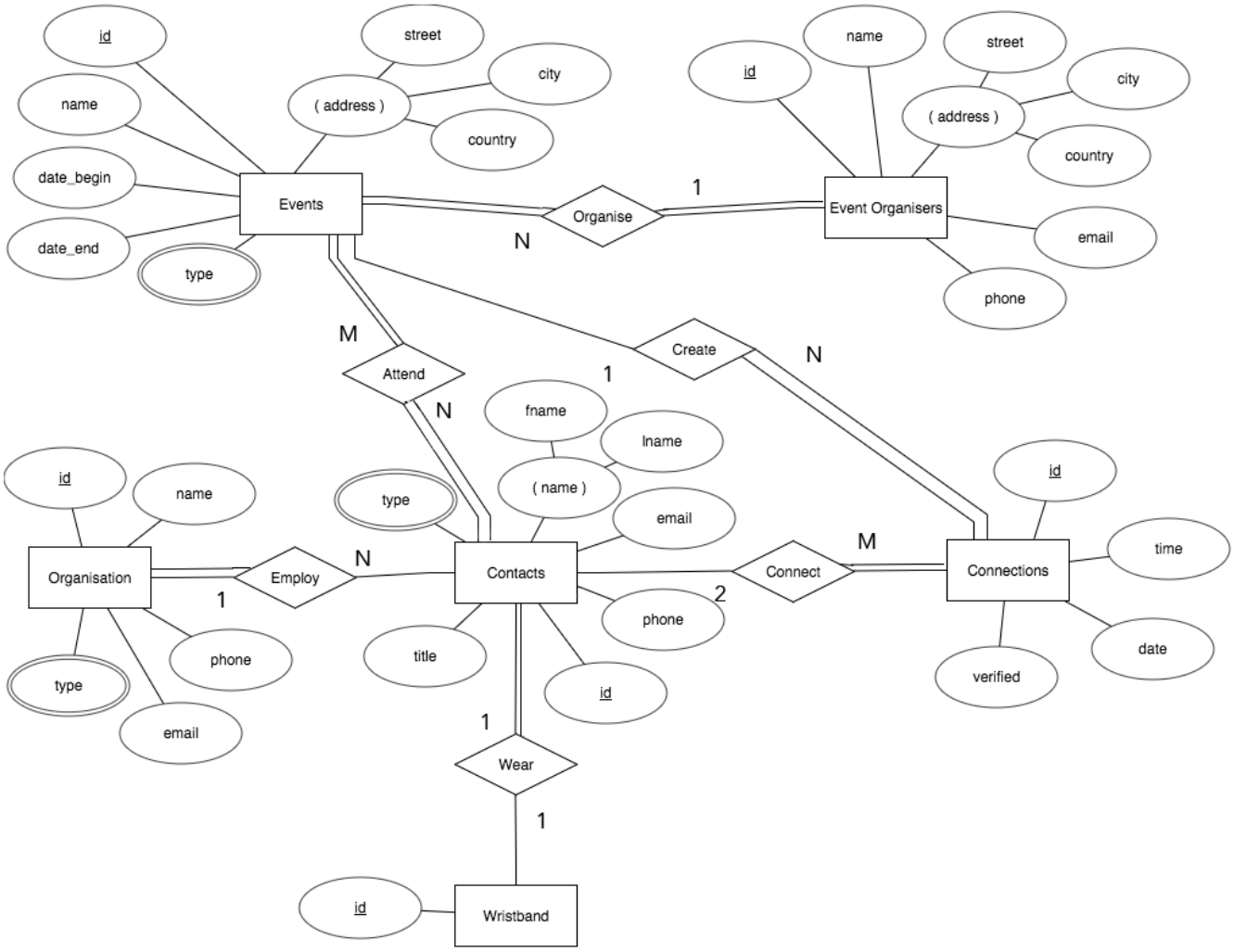
Explanation

This application involves the use of an electronic wristband at an event to automatically exchange the contact details of two people who shake hands while wearing wristbands. The idea is that this will remove the inefficiencies of business cards and facilitate a more seamless way to network at events by allowing the handshake to capture all of the person's contact information as well as the time, date and location of the meeting. In the following database, relationships are created between the event, event organiser, contacts, organisations of the contacts, wristbands that the contacts wear and most importantly the connections that occur between two contacts.

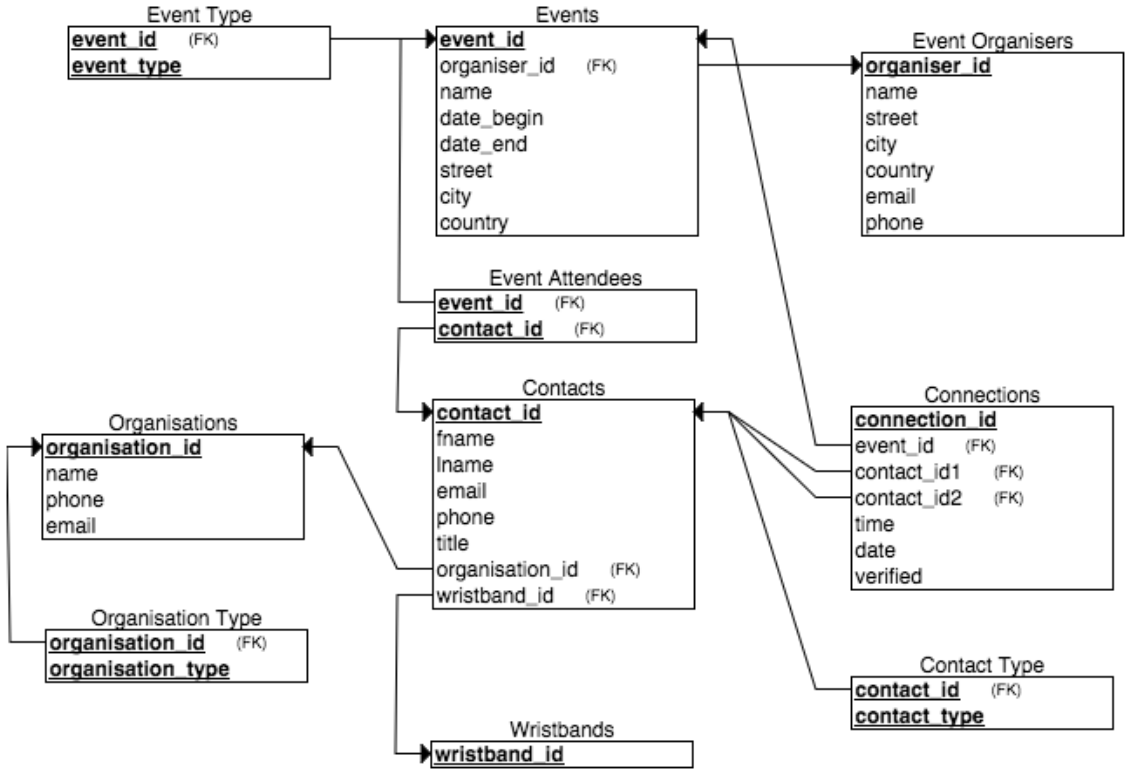
The wristbands will be distributed to every event attendee at the beginning of the event when they sign in to registration. Every event attendee will accordingly be issued a contact ID if they have not used the service before at a previous event (which can be checked by searching for their email address) and a corresponding wristband ID. Not every wristband will necessarily be used at all times, and it is important to have a surplus of wristbands in case some break or are lost, etc. Attendees will be encouraged to keep their wristbands for their next event (and their associated wristband ID). Information about the event attendee including their name, email, phone, title and organisation (the organisation is not necessary as an event attendee could be self-employed, etc.) will be entered at registration (this information could be acquired by using their email address and running it through an online customer acquisition service such as Full Contact to make the process more seamless).

Neglecting the finer details of how the handshake matching mechanism works and making the simple assumption that when two people shake hands, their wristband ID, the other person's wristband ID, the event ID and the date/time will reach the database for processing. This set of information is stored with the corresponding contact ID's to the wristband ID's and the other information acquired from the wristband. When another set of data reaches the database with the same wristband ID's but reversed in order (i.e. the person who is wearing the wristband will always have their wristband ID displayed first in the table) with the same event ID and a date/time that lie within a small error threshold of the first date/time, then both of the table entries become 'verified' and can now be confidently displayed to each event attendee as a connection that they have made by viewing it on their smart phone app, web site interface, etc.

Entity Relationship Diagram



Relational Schema



Functional Dependency Diagram

Semantic Constraints

The main semantic constraints for the database revolve around the Connections table, because a lot of its rules on verification cannot be directly expressed in the relational schema. The connection is only verified when two row entries are present in the Connections table with equal but inverted contact ID's, equal event ID, date and time. This semantic constraint is handled using a trigger as explained below and the 'verified' column is present in the table to indicate that the connection is ready to be displayed to the user.

Another semantic constraint is that the date of a connection should lie within the range of its event's start and end date. In a real-world use, an external application program would be responsible for getting the date correct and the event ID would be picked up by the server that processes the handshake over a wireless medium in a room in a particular location at the relevant event; therefore implementing this semantic constrain in SQL would add redundancy since the information needed is contained external to the database.

For the three different 'type' tables (EventType, OrganisationType and ContactType), a CHECK is used to ensure that the 'type' is limited to a finite set of possible values. E.g.

```
CONSTRAINT check_contact_type CHECK(contact_type IN
('Founder','Journalist','Investor','Other'))
```

One final semantic constraint that is not readily displayed in the relational schema is that the server does not receive the contact ID's from the wristbands, it instead receives the wristband ID's and performs a SELECT operation to find the relevant contact ID for insertion in a new row.

```
INSERT INTO Connections VALUES (1, 1, (SELECT contact_id FROM Contacts
WHERE wristband_id = 10), (SELECT contact_id FROM Contacts WHERE
wristband_id = 5), '13:04:21', '04-NOV-2016', DEFAULT);
```

Update Operations

Here are some topical update statements:

```
UPDATE Events SET event_city = 'Dublin', event_country = 'Ireland' WHERE
event_name = 'Web Summit';
UPDATE Events SET event_city = 'Lisbon', event_country = 'Portugal' WHERE
event_name = 'Web Summit';

UPDATE Contacts SET contact_title = 'Venture Capitalist' WHERE fname =
'Paul' AND lname = 'Hewson';
```

View Creation Using Table Joins Across Multiple Tables

Below is a view of the people that Bono (contact_id = 10) has shook hands with at two different events. This view combines information from four tables: Contacts, Connections, Events and Organisations and ensures the connections have been verified, which is explained below in the Trigger Command section. The view takes into account that this particular contact_id could be in either position of contact_id1 or contact_id2 depending on when it reached the server.

FNAME	LNAME	CONTACT_ EMAIL	CONTACT_ PHONE	CONTACT_ TITLE	ORGANISATION_ NAME	CONNECT_ TIME	CONNECT_ DATE	EVENT_N AME
Jack	Dorsey	jack@twitt er.com	234-578- 2346	Founder and CEO	Twitter	13:04:21	04-Nov-16	Web Summit
Mark	Zucker berg	zuck@face book.com	345-123- 6789	Founder and CEO	Facebook	11:10:02	17-Oct-16	SXSW

```
CREATE VIEW ContactInfo AS
SELECT Contacts.fname, Contacts.lname, Contacts.contact_email,
Contacts.contact_phone,
Contacts.contact_title, Organisations.organisation_name,
Connections.connect_time,
Connections.connect_date, Events.event_name
FROM Contacts, Connections, Events, Organisations
WHERE (Connections.contact_id1 = 10 OR Connections.contact_id2 = 10)
AND (Connections.contact_id1 = Contacts.contact_id OR
Connections.contact_id2 = Contacts.contact_id)
AND (Contacts.contact_id != 10)
AND Connections.event_id = Events.event_id
AND Contacts.organisation_id = Organisations.organisation_id
AND Connections.verified = 1;
```

Trigger Command

The following trigger command 'verifies' connections by deleting duplicated rows, i.e. each person involved in the handshake will push their connection packet to the server (with the contact ID's in reversed order) near the same time, and the packet that arrives second will remain while the first packet will be removed. In this project, data is inserted very simplistically, i.e. serially with one row after another in time and a problem is never encountered with the implementation below, however in a real world application, connection data will reach the server stochastically in time with the potential for two packets involving one connection to arrive at almost exactly the same time. This will cause a problem for the way this trigger is implemented below because two triggers could execute concurrently and not find the opposing contact ID and therefore not verify the connection. What could be done is to write to a special table when the trigger is firing and write again once it is finished. Before the trigger fires, it would check this table to ensure it is allowed to fire. This is somewhat inefficient, however one of the more inefficient alternatives would be to implement multiple tables for the connections (unmatched and matched) where the trigger would fire after an insert occurs by reading data from the unmatched connections table and writing verified transactions to the matched transactions table; two tables are used to avoid the problem of table mutation where a trigger attempts to write to a table that it has read from, which causes problems with reliability since the table may have changed since it has been read. The multiple tables method then increases the redundancy of the database by 3/2, and likely an extra step would be taken to delete data from the unmatched connections table after they are verified (which adds more inefficiency).

```
CREATE OR REPLACE TRIGGER verify_connection
BEFORE INSERT ON Connections
FOR EACH ROW
DECLARE
    new_contact_id1 integer;
    new_contact_id2 integer;
    new_event_id integer;
    new_connect_time varchar(255);
```

```
    new_connect_date date;
    i number;
BEGIN
    new_contact_id1 := :NEW.contact_id1;
    new_contact_id2 := :NEW.contact_id2;
    new_event_id := :NEW.event_id;
    new_connect_time := :NEW.connect_time;
    new_connect_date := :NEW.connect_date;
    DELETE FROM Connections WHERE contact_id1 = new_contact_id2 AND
contact_id2 = new_contact_id1
    AND event_id = new_event_id AND connect_time = new_connect_time AND
connect_date = new_connect_date;
    i := sql%rowcount;
    :NEW.verified := i;
END verify_connection;
.
RUN;
```

Security Policies

The security policy for this database is straightforward, it will have limited access from outside sources and be mainly self-contained to maintain its data and verify the connections. In the interest of maximising potential profit from selling this data to companies, no valid data will ever be deleted from the database (aside from the deletion of a row in the Connections table that has been verified), and therefore no user is given DELETE privilege.

In this example, an account administrator is granted access to view and update data specifically related to Event Organisers, who are the customers of the handshaking app. The account administrator can also grant privilege to others to only view the Event Organisers tables (e.g. a colleague at work).

```
GRANT SELECT ON Events TO some_account_admin WITH GRANT OPTION;
GRANT SELECT ON EventType TO some_account_admin WITH GRANT OPTION;
GRANT SELECT ON EventOrganisers TO some_account_admin WITH GRANT OPTION;
GRANT UPDATE ON Events TO some_account_admin;
GRANT UPDATE ON EventType TO some_account_admin;
GRANT UPDATE ON EventOrganisers TO some_account_admin;
```

These privileges could also be encapsulated within what is called a ‘role’, where privileges are granted to this specific role and the role is then granted to a user at a later stage.

```
CREATE ROLE Account_Admin_Role IDENTIFIED BY aar;
GRANT SELECT ON Events TO Account_Admin_Role WITH GRANT OPTION;
GRANT SELECT ON EventType TO Account_Admin_Role WITH GRANT OPTION;
GRANT SELECT ON EventOrganisers TO Account_Admin_Role WITH GRANT OPTION;
GRANT UPDATE ON Events TO Account_Admin_Role;
GRANT UPDATE ON EventType TO Account_Admin_Role;
GRANT UPDATE ON EventOrganisers TO Account_Admin_Role;

GRANT Account_Admin_Role TO some_account_admin1;
GRANT Account_Admin_Role TO some_account_admin2;
```

The following example is of a server/router that is located at an event and is picking up encoded signals from the physical wristbands over a wireless medium. For the duration of the particular event, the server is granted access to make insertions to the Connections table of the database, and as soon as the event ends, this privilege is revoked.

```
GRANT INSERT ON Connections TO some_event_server;
```

```
REVOKE INSERT ON Connections TO some_event_server;
```

The following is a granting of privilege to a particular event attendee (in this case every event attendee) to have access to the View table of the connections that they have personally made (this table is shown above).

```
GRANT SELECT ON ContactInfo TO some_event_attendee;
```

Code Appendix

Create Tables

```
CREATE TABLE Wristbands
(
wristband_id int NOT NULL,
PRIMARY KEY(wristband_id)
);

CREATE TABLE Organisations
(
organisation_id int NOT NULL,
organisation_name varchar(255) NOT NULL,
organisation_phone varchar(255) NOT NULL,
organisation_email varchar(255) NOT NULL,
PRIMARY KEY(organisation_id)
);

CREATE TABLE OrganisationType
(
organisation_id int NOT NULL,
organisation_type varchar(255) NOT NULL,
PRIMARY KEY(organisation_id, organisation_type),
FOREIGN KEY(organisation_id) REFERENCES Organisations(organisation_id),
CONSTRAINT check_organisation_type CHECK(organisation_type IN
('Tech','Finance','Media','Other'))
);

CREATE TABLE EventOrganisers
(
organiser_id int NOT NULL,
organiser_name varchar(255) NOT NULL,
organiser_street varchar(255) NOT NULL,
organiser_city varchar(255) NOT NULL,
organiser_country varchar(255) NOT NULL,
organiser_email varchar(255) NOT NULL,
organiser_phone varchar(255) NOT NULL,
PRIMARY KEY(organiser_id)
);

CREATE TABLE Events
(
event_id int NOT NULL,
organiser_id int NOT NULL,
event_name varchar(255) NOT NULL,
date_begin date NOT NULL,
date_end date NOT NULL,
event_street varchar(255) NOT NULL,
event_city varchar(255) NOT NULL,
event_country varchar(255) NOT NULL,
PRIMARY KEY(event_id),
FOREIGN KEY(organiser_id) REFERENCES EventOrganisers(organiser_id)
```

```
);

CREATE TABLE EventType
(
event_id int NOT NULL,
event_type varchar(255) NOT NULL,
PRIMARY KEY(event_id, event_type),
FOREIGN KEY(event_id) REFERENCES Events(event_id),
CONSTRAINT check_event_type CHECK(event_type IN
('Tech','Careers','Business','Other'))
);

CREATE TABLE Contacts
(
contact_id int NOT NULL,
fname varchar(255) NOT NULL,
lname varchar(255) NOT NULL,
contact_email varchar(255) NOT NULL,
contact_phone varchar(255) NOT NULL,
contact_title varchar(255) NOT NULL,
organisation_id int,
wristband_id int NOT NULL,
PRIMARY KEY(contact_id),
FOREIGN KEY(organisation_id) REFERENCES Organisations(organisation_id),
FOREIGN KEY(wristband_id) REFERENCES Wristbands (wristband_id)
);

CREATE TABLE ContactType
(
contact_id int NOT NULL,
contact_type varchar(255) NOT NULL,
PRIMARY KEY(contact_id, contact_type),
FOREIGN KEY(contact_id) REFERENCES Contacts(contact_id),
CONSTRAINT check_contact_type CHECK(contact_type IN
('Founder','Journalist','Investor','Other'))
);

CREATE TABLE EventAttendees
(
event_id int NOT NULL,
contact_id int NOT NULL,
PRIMARY KEY(event_id, contact_id),
FOREIGN KEY(event_id) REFERENCES Events(event_id),
FOREIGN KEY(contact_id) REFERENCES Contacts(contact_id)
);

CREATE TABLE Connections
(
connection_id int NOT NULL,
event_id int NOT NULL,
contact_id1 int NOT NULL,
contact_id2 int NOT NULL,
connect_time varchar(255) NOT NULL,
connect_date date NOT NULL,
verified int DEFAULT 0 NOT NULL,
PRIMARY KEY(connection_id),
FOREIGN KEY(event_id) REFERENCES Events(event_id),
FOREIGN KEY(contact_id1) REFERENCES Contacts(contact_id),
FOREIGN KEY(contact_id2) REFERENCES Contacts(contact_id)
);
```

Insert Into Tables

```
INSERT INTO Wristbands VALUES (1);
INSERT INTO Wristbands VALUES (2);
INSERT INTO Wristbands VALUES (3);
INSERT INTO Wristbands VALUES (4);
INSERT INTO Wristbands VALUES (5);
INSERT INTO Wristbands VALUES (6);
INSERT INTO Wristbands VALUES (7);
INSERT INTO Wristbands VALUES (8);
INSERT INTO Wristbands VALUES (9);
INSERT INTO Wristbands VALUES (10);
INSERT INTO Wristbands VALUES (11);
INSERT INTO Wristbands VALUES (12);

INSERT INTO Organisations VALUES (1, 'Facebook', '111-111-1111',
'facebook@facebook.com');
INSERT INTO Organisations VALUES (2, 'Delta Partners', '222-222-2222',
'dp@dp.com');
INSERT INTO Organisations VALUES (3, 'Twitter', '333-333-3333',
'twitter@twitter.com');
INSERT INTO Organisations VALUES (4, 'KPMG', '444-444-4444',
'kpmg@kpmg.com');
INSERT INTO Organisations VALUES (5, 'Wall Street Journal', '555-555-5555',
'wsj@wsj.com');

INSERT INTO OrganisationType VALUES (1, 'Tech');
INSERT INTO OrganisationType VALUES (2, 'Tech');
INSERT INTO OrganisationType VALUES (3, 'Tech');
INSERT INTO OrganisationType VALUES (4, 'Finance');
INSERT INTO OrganisationType VALUES (5, 'Media');

INSERT INTO EventOrganisers VALUES (1, 'Ci', '1 Main Street', 'Lisbon',
'Portugal', 'ci@ci.com', '123-456-7890');
INSERT INTO EventOrganisers VALUES (2, 'SXSW Ltd.', '2 Main Street',
'Austin', 'USA', 'sxsw@sxsw.com', '234-567-8901');
INSERT INTO EventOrganisers VALUES (3, 'Burning Man Events', '3 Main
Street', 'Reno', 'USA', 'bm@bm.com', '345-678-9012');
INSERT INTO EventOrganisers VALUES (4, 'SAP Ltd.', '4 Main Street',
'Chicago', 'USA', 'sap@sap.com', '456-789-0123');
INSERT INTO EventOrganisers VALUES (5, 'Dublin City Council', '5 Main
Street', 'Dublin', 'Ireland', 'dcs@dcs.com', '567-890-1234');

INSERT INTO Events VALUES (1, 1, 'Web Summit', '04-NOV-2016', '07-NOV-
2016', '11 Main Street', 'Lisbon', 'Portugal');
INSERT INTO Events VALUES (2, 1, 'Collision', '06-JUN-2016', '09-JUN-2016',
'21 Main Street', 'Las Vegas', 'USA');
INSERT INTO Events VALUES (3, 2, 'SXSW', '15-OCT-2016', '20-OCT-2016', '32
Main Street', 'Austin', 'USA');
INSERT INTO Events VALUES (4, 3, 'Burning Man', '21-JUL-2016', '24-JUL-
2016', '43 Main Street', 'Reno', 'USA');
INSERT INTO Events VALUES (5, 5, 'Careers Fair', '11-NOV-2016', '15-NOV-
2016', '55 Main Street', 'Dublin', 'Ireland');

INSERT INTO EventType VALUES (1, 'Tech');
INSERT INTO EventType VALUES (1, 'Business');
INSERT INTO EventType VALUES (2, 'Tech');
INSERT INTO EventType VALUES (2, 'Business');
INSERT INTO EventType VALUES (3, 'Tech');
INSERT INTO EventType VALUES (3, 'Other');
INSERT INTO EventType VALUES (4, 'Other');
INSERT INTO EventType VALUES (5, 'Careers');
```

```
INSERT INTO Contacts VALUES (1, 'John', 'Smith', 'john.smith@facebook.com',
'134-436-3773', 'Software Engineer', 1, 1);
INSERT INTO Contacts VALUES (2, 'Rebecca', 'Appleby',
'rebecca.appleby@facebook.com', '345-245-4678', 'Systems Engineer', 1, 2);
INSERT INTO Contacts VALUES (3, 'Adam', 'Craigson', 'adam.craigson@dp.com',
'389-284-3467', 'Venture Capitalist', 2, 3);
INSERT INTO Contacts VALUES (4, 'Michael', 'Adams', 'michael.adams@dp.com',
'234-234-4568', 'Venture Capitalist', 2, 4);
INSERT INTO Contacts VALUES (5, 'Jack', 'Dorsey', 'jack@twitter.com', '234-
578-2346', 'Founder and CEO', 3, 5);
INSERT INTO Contacts VALUES (6, 'Mark', 'Zuckerberg', 'zuck@facebook.com',
'345-123-6789', 'Founder and CEO', 1, 6);
INSERT INTO Contacts VALUES (7, 'Lisa', 'Fleisher',
'lisa.fleisher@wsj.com', '948-582-4057', 'Tech Journalist', 5, 7);
INSERT INTO Contacts VALUES (8, 'Joe', 'Mahony', 'joe.mahony@kpmg.com',
'248-274-4235', 'Account Specialist', 4, 8);
INSERT INTO Contacts VALUES (9, 'Mark', 'Strongman',
'mark.strongman@wsj.com', '235-346-3245', 'Finance Journalist', 5, 9);
INSERT INTO Contacts VALUES (10, 'Paul', 'Hewson', 'bono@u2.com', '234-356-
1235', 'Singer', NULL, 10);

INSERT INTO ContactType VALUES (1, 'Other');
INSERT INTO ContactType VALUES (2, 'Other');
INSERT INTO ContactType VALUES (3, 'Investor');
INSERT INTO ContactType VALUES (4, 'Investor');
INSERT INTO ContactType VALUES (5, 'Founder');
INSERT INTO ContactType VALUES (5, 'Investor');
INSERT INTO ContactType VALUES (6, 'Founder');
INSERT INTO ContactType VALUES (6, 'Investor');
INSERT INTO ContactType VALUES (7, 'Journalist');
INSERT INTO ContactType VALUES (8, 'Other');
INSERT INTO ContactType VALUES (9, 'Journalist');
INSERT INTO ContactType VALUES (10, 'Founder');
INSERT INTO ContactType VALUES (10, 'Investor');
INSERT INTO ContactType VALUES (10, 'Other');

INSERT INTO EventAttendees VALUES (1, 5);
INSERT INTO EventAttendees VALUES (1, 10);
INSERT INTO EventAttendees VALUES (2, 1);
INSERT INTO EventAttendees VALUES (2, 2);
INSERT INTO EventAttendees VALUES (3, 10);
INSERT INTO EventAttendees VALUES (4, 6);
INSERT INTO EventAttendees VALUES (5, 3);
INSERT INTO EventAttendees VALUES (1, 4);
INSERT INTO EventAttendees VALUES (2, 7);
INSERT INTO EventAttendees VALUES (3, 8);
INSERT INTO EventAttendees VALUES (3, 9);

INSERT INTO Connections VALUES (1, 1, (SELECT contact_id FROM Contacts
WHERE wristband_id = 10), (SELECT contact_id FROM Contacts WHERE
wristband_id = 5), '13:04:21', '04-NOV-2016', DEFAULT);
INSERT INTO Connections VALUES (2, 1, (SELECT contact_id FROM Contacts
WHERE wristband_id = 5), (SELECT contact_id FROM Contacts WHERE
wristband_id = 10), '13:04:21', '04-NOV-2016', DEFAULT);
INSERT INTO Connections VALUES (3, 2, (SELECT contact_id FROM Contacts
WHERE wristband_id = 1), (SELECT contact_id FROM Contacts WHERE
wristband_id = 2), '15:14:32', '07-JUN-2016', DEFAULT);
INSERT INTO Connections VALUES (4, 2, (SELECT contact_id FROM Contacts
WHERE wristband_id = 2), (SELECT contact_id FROM Contacts WHERE
wristband_id = 1), '15:14:32', '07-JUN-2016', DEFAULT);
INSERT INTO Connections VALUES (5, 3, (SELECT contact_id FROM Contacts
WHERE wristband_id = 10), (SELECT contact_id FROM Contacts WHERE
wristband_id = 6), '11:10:02', '17-OCT-2016', DEFAULT);
```

```
INSERT INTO Connections VALUES (6, 3, (SELECT contact_id FROM Contacts
WHERE wristband_id = 6), (SELECT contact_id FROM Contacts WHERE
wristband_id = 10), '11:10:02', '17-OCT-2016', DEFAULT);
```