1. Using lab 3 Uver as an example to show a **data model using the Entity-Relationship model**

There are 2 major components in making an ER model. First are the entities which is the thing or object with independent existence e.g., customer, ride, driver, car. Second, is the relationship which is how entities interact or refer to each other e.g., a driver has a car, a customer occupies a ride. A property of an entity is shows as an oval, for example customer entity has a "name" property, and attributes are attached to their entity with a straight line. A key attribute is a primary key where it uniquely identifies an entity, these are underlined in and ERD. All entities must have at least one key. A composite attribute represents some property that is compose of parts. For example, a location attribute has a point and an address, these are represented by an oval and are connected to its attribute. A multivalued represents something that can have multiple entries. For example, a feature attribute of the car entity has roof rack, pet seat, AC, Wi-Fi. This is represented with a double oval. A derived attribute is a value that can be derived from other attributes, in other words, this does not need to be stored. For example, distance from the ride entity can be found by subtracting location from customer entity and destination in ride entity. This is represented by dashed outline. A relationship is represented as a diamond and they must attach two or more entities. Connecters have cardinality ratios where there are a number of instances of each entity that participates in the relationship with an instance on the other side. For example, car has a 1:1 relation with driver since only one car can be driven by one driver, and customer has a N:M relationship with rides since many customers can occupy many rides. A weak entity depends for its existence and identity on a relationship with some other entity. A weak entity only has a partial key. For example, ride is a weak entity since it depends on the car entity and customer entity for its existence. From here, pickup_time is its partial key. A weak entity is represented had double-board rectangle, an identifying relation of the weak key is a double-boarded diamond, a partial key is a dashed underline. A relationship can also have an attribute which represents values but don't belong to wither entity. These are also represented as an oval just like a regular attribute. The diagram below shows a complete model based on the examples mentioned above.[1]

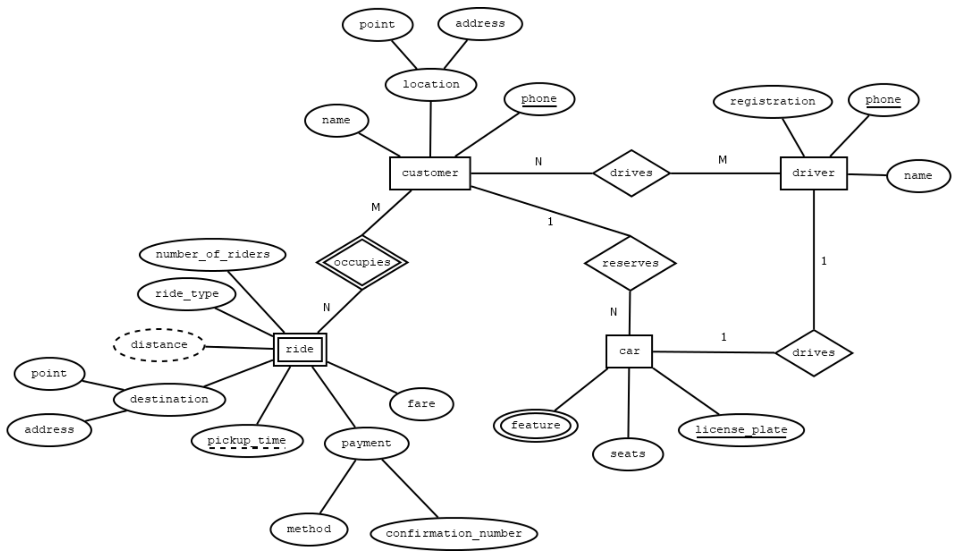


Figure 1: From lab 3 Uver. Denisha Saviela diagram

2. Using lab 3 Uver to **design a relational database schema implementing a data model**

In general, entities are tables, 1:1 and 1: N relationships are foreign keys, N:M relationships are tables, and multivalued attributes are tables. We can follow 7-step algorithm to design a relational database schema implementing a data model.[2]

Step 1: Entities are tables and attributes are columns. Choose some key to be a primary key.	`CREATE TABLE customer(`
	`   name TEXT UNIQUE,`
	`   location_point POINT,`
	`   location_address TEXT,`
	`   phone VARCHAR(12) PRIMARY KEY);`

Step 2: Weak entities take a primary key from owning entity's table add it to its weak entity table and make it a foreign key back to owning entity's table. Make a primary key a combination of a partial key and "borrowed" key from owning entity.	`CREATE TABLE ride(`
	`   customer_name TEXT REFERENCES customer(name),`
	`   number_of_riders INTEGER,`
	`   ride_type TEXT,`
	`   distance TEXT,`
	`   destination_point POINT,`
	`   destination_address TEXT,`
	`   fare NUMERIC(10,2),`
	`   pickup_time TIMESTAMP,`
	`   payment_method TEXT,`
	`   payment_confirmation_number CHAR(11) UNIQUE,`

[1] Database Modeling with Entity-Relationship Diagrams 2/6/2023
[2] Convert ERD to SQL 2/10/2023

	`PRIMARY KEY (customer_name, pickup_time));`
Step 3: 1:1 relationship, total participation on both sides can merge into one table	`-- drives relationship` `ALTER TABLE driver` `ADD COLUMN car_license_plate CHAR(10)` `REFERENCES car(license_plate);`
Step 4: 1: N relationship can take primary key from “one” side and add to “many” side table.	`ALTER TABLE car` `ADD COLUMN customer_phone VARCHAR(12)` `REFERENCES customer(phone);`
Step 5: N:M relationship creates a new table whose entries represent connections between entries in the two tables. The new table will borrow the primary key from each table, each borrowed key is a foreign key back to original. For short, we will use xref to describe its table name.	`CREATE TABLE customer_driver_xref(` `phone VARCHAR(12),` `driver_phone VARCHAR(12) REFERENCES driver(phone),` `PRIMARY KEY (phone, driver_phone),` `FOREIGN KEY (phone)` `REFERENCES customer(phone));`
Step 6: Multivalued attributes can essentially be treated like a weak entity. Make a table for the attribute, in place of partial key, use the attribute, borrow a primary key of an owning table, make the borrowed key and attribute together the primary key, and make a foreign key back to its owning table.	`CREATE TABLE car_feature(` `car_license_plate TEXT,` `feature TEXT,` `PRIMARY KEY (car_license_plate, feature),` `FOREIGN KEY (car_license_plate)` `REFERENCES car(license_plate));`
Step 7: N-ary relationships use a cross-reference table. As for N:M relationships, but using primary keys from all involved tables.	
While derived attributes is not part of the 7-step algorithm, we can still create one for this example. We can create a view that encompasses the derived attribute. This will contain all the attribute of the original entity plus the derived attribute.	`CREATE VIEW ride_plus AS` `SELECT car_license_plate,` `number_of_riders,` `ride_type,` `distance,` `destination_address,` `fare,` `pickup_time,` `payment_method,` `payment_confirmation_number,` `destination_point<@>location_point AS destination_distance` `FROM ride r join customer c` `ON c.name = r.customer_name;`

3. Using lab 2 Schedule as an example we can **create database objects and upload data to populate a database; and issue statements to add, remove, and alter data and metadata in a database**[3]

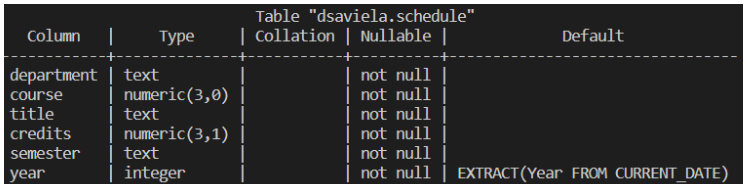
To insert data into tables, the number and types of values must match the number of and the types of the specified columns. Any column not specified gets NULL, unless there is a default, and if you omit the columns list, then SQL will assume you are providing values for all columns in order. For example, we can create a schedule table using the script below:

```
CREATE TABLE schedule (

    department text,

    course numeric(3),

    title text NOT NULL,
```



```
                     Table "dsaviela.schedule"
   Column   |     Type     | Collation | Nullable |            Default
------------+--------------+-----------+----------+--------------------------------
 department | text         |           | not null |
 course     | numeric(3,0) |           | not null |
 title      | text         |           | not null |
 credits    | numeric(3,1) |           | not null |
 semester   | text         |           | not null |
 year       | integer      |           | not null | EXTRACT(Year FROM CURRENT_DATE)
```

[3] DDL and DML CREATE TABLE and INSERT/UPDATE/DELETE 1/27/2023

```
    credits numeric(3,1) NOT NULL,

    semester text CHECK (semester IN ( 'Spring', 'Fall', 'Summer')),

    year int DEFAULT EXTRACT('Year' FROM CURRENT_DATE),

    PRIMARY KEY (department, course));
```

We can use an INSERT INTO.. VALUES such as

```
INSERT INTO schedule VALUES ('MATH', 201, 'INTRO TO STATISTICS', 3.0, 'Spring');
```

A shorthand way to get data from one table to another is to use INSERT INTO … SELECT

```
INSERT INTO schedule

    SELECT department, course_number, course_title, semester_hours,

    FROM cs_courses

    WHERE

        course_number IN (303, 306, 403, 406);
```

We can also use DELETE where it deletes rows matching the, optional, WHERE clause:

```
DELETE FROM schedule

WHERE course = 303;
```

We can fix potential errors and add more constraint using UPDATE where it modifies only rows matching and ALTER TABLE:

```
UPDATE schedule

SET title = 'DATABASE MANAGEMENT'

WHERE course = 403;
```

```
ALTER TABLE schedule

ADD CONSTRAINT credits_check

CHECK(credits BETWEEN 0.5 AND 15);
```

4. Using subqueries and JOIN to **construct retrieval and data modification queries in SQL**

In SQL, to retrieve data stored in our tables, we use the SELECT statement. Queries can return a list of values, or tuples, and we can substitute a query for a list. For example, the script below finds books by living authors:

```
SELECT title, author

FROM books

WHERE author IN

    (SELECT name

    FROM authors

    WHERE death IS NULL);
```

Subquery queries are often equivalent to join queries, when we only want columns from the table involved in the outer query. For example, the script below finds posthumous books:

```
SELECT title, author

FROM books b

WHERE publication_year > (SELECT EXTRACT(year FROM death)
        FROM authors a

        WHERE b.authors = a.name);
```

A subquery can return multiple things, such as a table, a single tuple, a scalar value, and nothing. With single tuple, these subqueries can be used in equality comparison expressions. The example above falls under this category. For non-scalar tuple, if you have multiple value tuples, use parentheses. Scalar values can be used in any expression and does not have to be in WHERE. For example, the script below finds the person who made the most sales in February, how many sales they made, and the total sum. This example also uses correlates subquery, where the subquery accesses attributes from rows in the outer query (s2).4

```
SELECT salesperson, COUNT(*), SUM(amount)

FROM sales s

WHERE EXTRACT(month FROM order_date) = 2

GROUP BY salesperson

HAVING COUNT(*) =

    (SELECT MAX(count)

        FROM (SELECT COUNT(*) AS count FROM sales s2

            WHERE EXTRACT(month FROM order_date) = 2

            GROUP BY salesperson) x );
```

There are two ways to use a JOIN, in the WHERE clause implicit joins and FROM clause explicit joins. JOINS can have multiple conditions and multiple tables. INNER joins is the default and the performance is the same using the WHERE clause. They only return rows that match join condition. An outer join allows to get everything, pairing up rows where possible. There are also LEFT and RIGHT joins, where RIGHT changes which table, we take all rows from. To get all rows from both tables in join, you can use FULL OUTR JOIN, matching rows where possible. For example, the script below generates a list of all courses and faculty, showing course/faculty data together, but including all courses and all faculty, even courses without faculty and faculty without courses:

```
SELECT c.couse_id, c.instructor, f.name

FROM mines_cs_courses c FULL JOIN mines_cs_faculty f ON c.instructor = f.name
```

Outer joins are helpful for finding missing data, using IS NULL in the WHERE clause. For example, from the music schema, the script below finds artist with no members (solo artist):

```
SELECT a.name, a.type

FROM artist AS a

LEFT JOIN artist_member_xref AS x

ON (a.id = x.artist_id)

WHERE x.artist_id IS NULL;
```

[4] Subqueries 3/3/2023