DWI LAB – Mini-Project II

IMPLEMENTATION STAGE

1. ETL implementation

a. Define a general overview of the process – plan and architecture

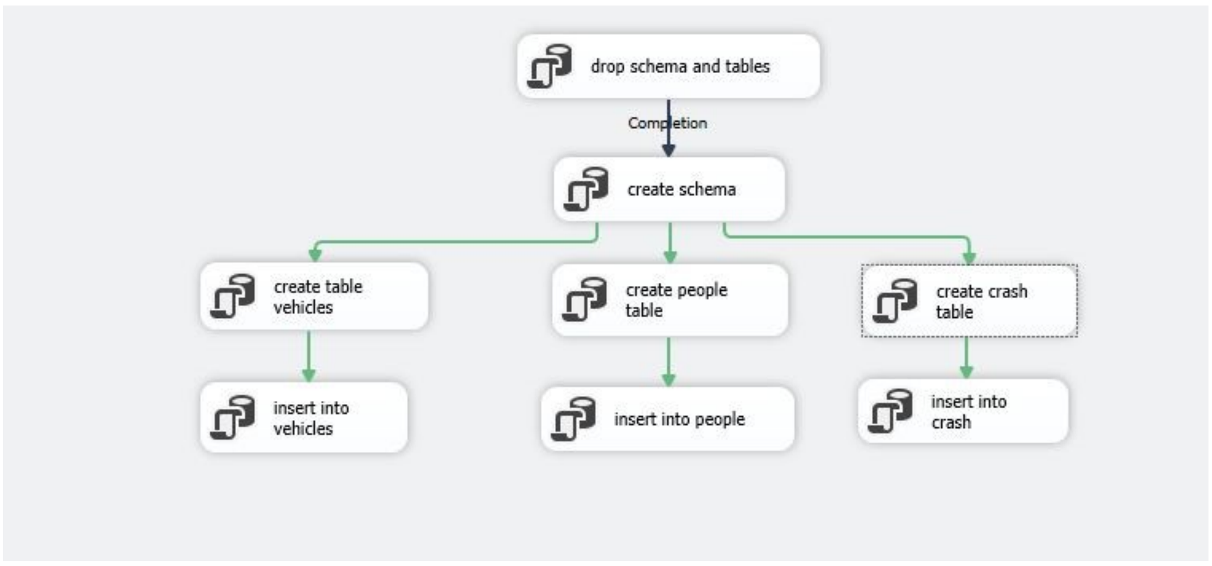
We have used SSIS Designer to create SQL Server Integration Services packages. The packages we create imports data from 3 flat files which are the main sources for the raw data, extracts the data the respective tables, reformats the data, performs some data profiling and then inserts the re-formatted data into our respective tables.

b. Perform extraction of needed data

i. Identify required extraction approach and implement it

After downloading our selected data(people, crash and vehicles) from the source website in csv format, we make three packages for obtaining data from our flat files and inserting this data from source to tables. *(peopleStore, crashStore, vehicleStore) using an OLE DB provider.*

Since we are not using all the columns as in the raw source, we make another package executing SQL tasks to create new tables (*PeopleUsefulData* , *CrashUsefulData*, *VehicleUsefulData*) and then inserting the required data to them from our initial tables.

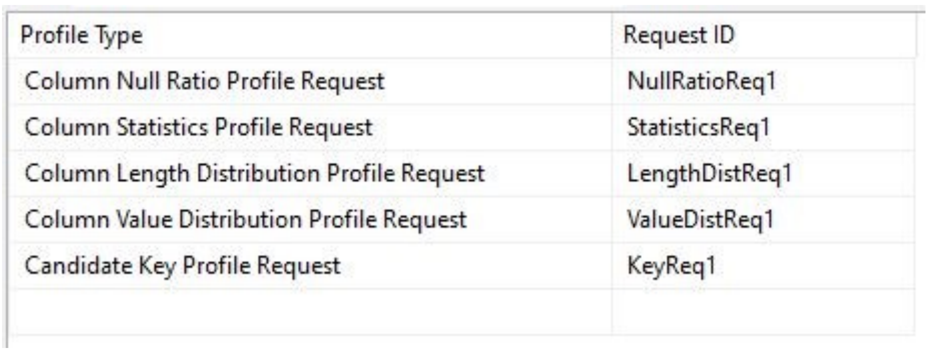


The picture above shows the SSIS package with the SQL instructions related to the creation of the three tables containing only the useful data for our project and the insertion of proper data into them. These tables will be the basis for the following phases: data profiling and anomaly detection will be performed on them, then data will be transformed according to the results.

c. Perform basic anomaly detection and resolution transformations

i. Define a basic set of data-quality screens

To ensure the quality of the data, at first we used the Data Profiling task in SSIS to perform basic analysis on the data inside the three tables *PeopleUsefulData* , *CrashUsefulData*, *VehicleUsefulData* .
The following profile requests were made:



Profile Type	Request ID
Column Null Ratio Profile Request	NullRatioReq1
Column Statistics Profile Request	StatisticsReq1
Column Length Distribution Profile Request	LengthDistReq1
Column Value Distribution Profile Request	ValueDistReq1
Candidate Key Profile Request	KeyReq1

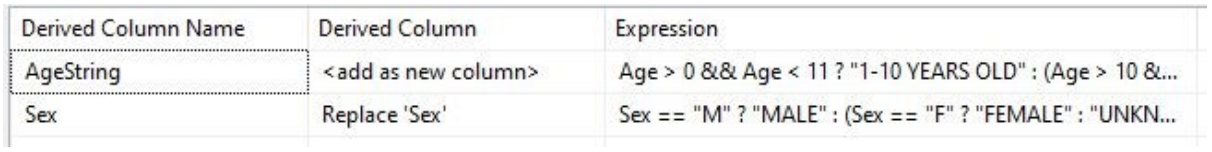
Then we analysed more specifically certain delicate attributes, such as those of integer types. In particular we searched for:

- Integer attributes with negative values;
- Integer attributes with unreasonable values (such as VehicleYear > 2020 or Age > 120);
- String attributes left empty; - NULL values.

ii. Clean the identified anomalies

The following picture shows the transformation and cleaning process operated on *PeopleUsefulData*, *CrashUsefulData* and *VehicleUsefulData* according to the previous analysis. In particular, we created a Data Flow activity for each of the three table and we used the Derived Column functionality. In particular, for *PeopleUsefulData* we:

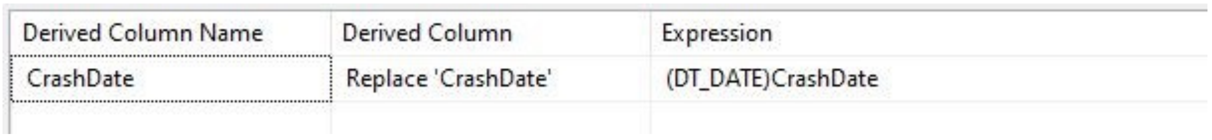
- changed Sex to UNKNOWN any time no value is given or a value different from M and F is given; M and F are made verbose;
- grouped together Age into decades described by a String value.



Derived Column Name	Derived Column	Expression
AgeString	<add as new column>	Age > 0 && Age < 11 ? "1-10 YEARS OLD" : (Age > 10 &...
Sex	Replace 'Sex'	Sex == "M" ? "MALE" : (Sex == "F" ? "FEMALE" : "UNKN...

For *CrashUsefulData* we:

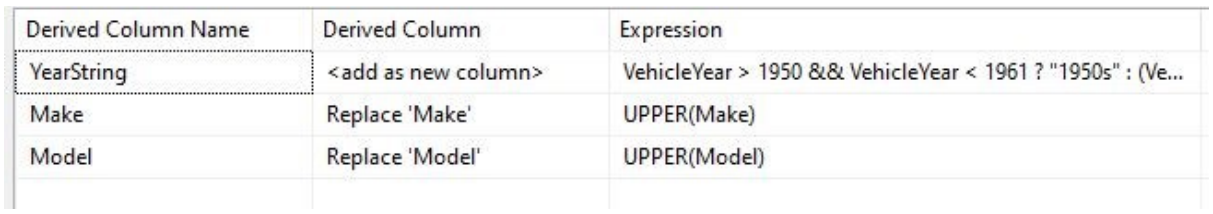
- changed the type of the attribute CrashDate.



Derived Column Name	Derived Column	Expression
CrashDate	Replace 'CrashDate'	(DT_DATE)CrashDate

For *VehicleUsefulData* we:

- capitalised both the attributes Make and Model, to standardise them with the rest of the data;
- grouped together VehicleYear to have more significant values, since the number of old cars is a lot lower in respect to the number of modern cars, so we grouped those in decades described by a String value.



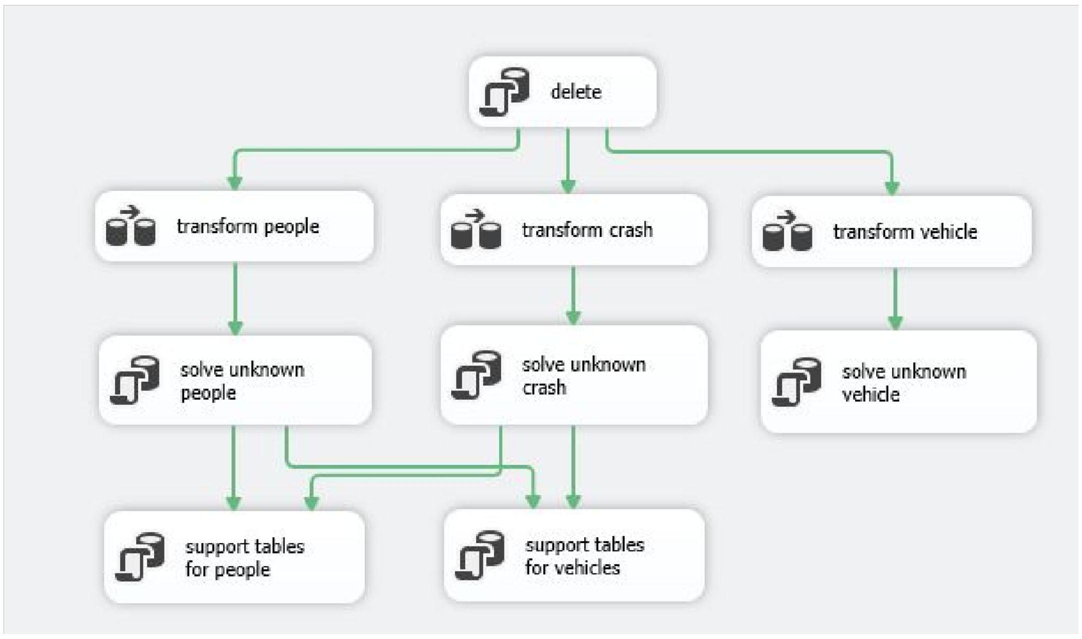
Derived Column Name	Derived Column	Expression
YearString	<add as new column>	VehicleYear > 1950 && VehicleYear < 1961 ? "1950s" : (Ve...
Make	Replace 'Make'	UPPER(Make)
Model	Replace 'Model'	UPPER(Model)

After this, we updated the tables to fix some anomalies we previously identified. In particular we eliminated the NULL (or blank) values in the following attributes:

- Maneuver
- UnitType
- Make
- Model
- SafetyEquipment
- AirbagDeployed
- InjuryClassification
- PhysicalCondition

Changing it to UKNOWN or similar, according to the value already in the data.

The following picture illustrates the process described before. The two last SQL commands create support tables for the creation of the Bridge Tables in the following point.



d. Perform needed data transformation and creation of assumed multi-dimensional scheme - star, snowflake, or fact constellation:

i. Schema creation and general design

We named our schema *crash* and we designed a star schema for our data warehouse.

ii. Dimension tables

We created the following dimension tables: *Date* , *Environment* , *Location* , *VehicleStatus* and *PeopleStatus* .

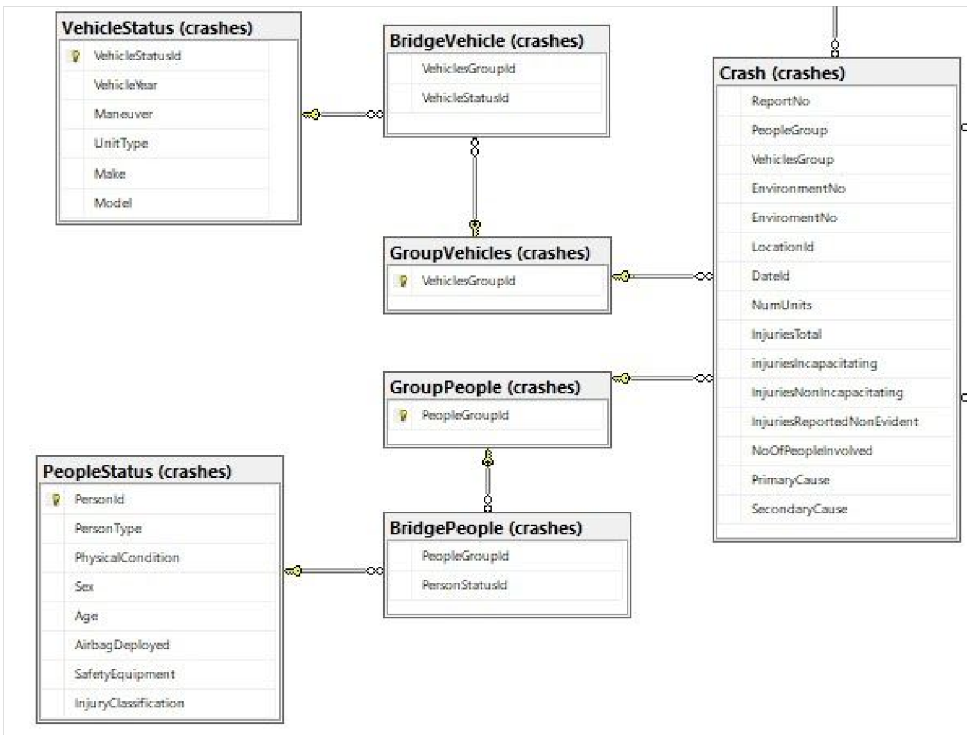
Each dimension has a progressive integer identifier as primary key, created with IDENTITY(1,1) function at the time the creation of the table. Only *Date* has a type date as a primary key for the table, but an additional integer surrogate key is made to link the dimension to the fact table.

The names of the attributes were already verbose and understandable, but all of them were standardised: first letter of each word capitalised and the rest lowercase (for instance TRAFFIC_CONTROL_DEVICE renamed into TrafficControlDevice).

The two dimensions *VehicleStatus* and *PeopleStatus* have been created grouping together different possible statuses of respectively units/vehicles and people involved in the crash. Statuses were extracted from the data sources and each row of the dimensions table was assigned an integer identifier for that particular condition.
To make the data more meaningful in these two tables, we grouped together values regarding the age of the people and the year the vehicle was made (for instance, all the people aged from 11 to 20 belong to the same age group). We made this inside the Data Flow activities shown in the previous point.

iii. Bridge tables

We designed two bridge tables for our model: *BridgePeople* and *BridgeVehicle* . Those are connected to two group dimensions (respectively *GroupPeople* and *GroupVehicles*) intended to be connected to the central fact table as shown in the following picture:



To create this structure, the following procedure was adopted, for data regarding both people and vehicles:

- create a support table linking crashes (identified by their unique report number) to the identifier (taken from the corresponding dimension) of the status of each person/vehicle involved in the crash - each row contains a report number and a person/vehicle ID - more rows are possible for the same report number;
- create a support table linking crashes (identified by their unique report number) to the list of people/vehicles involved in the crash, representing a group - one row for each report number;
- create a support table that assign a progressive unique integer identifier (*GroupID*) to each group (i.e. list) of people/vehicles involved in the crash present in the data sources.

Now, joining all these three support tables together, we could link crashes, groups of people/vehicles involved, identifiers of these groups, individual people/vehicles statuses.

GroupPeople and *GroupVehicles* were populated with distinct values of *GroupID* for people and vehicles respectively.

BridgePeople and *BridgeVehicle* were populated with distinct values of *GroupID* and *VehicleStatusID/PeopleStatusID*.

iv. Create fact table

We created a fact table *Crash* having FKs to implement connection to the dims(EnvironmentNo, Dateid,LocationID) and to the bridge tables(VehicleGroupID, PeopleGroupID). We have the following measures inside it: NumUnits, InjuriesTotal, InjuriesFatal, InjuriesInapacitating, InjuriesNonIncapapacitating, NoOfPeopleInvolved, InjuriesReportedNonEvident, all of type int.

To insert data into our fact table, after selecting the column names, we use the first left join for [crashes].[Date] on CrashDate. Second left join for [crashes].Environment on TrafficDevice, DeviceCondition, WeatherCondition, LightingCondition, RoadDefect, Alignment and RoadSurfaceCondition.

Another left join for [crashes].[Location] on BeatOfOccurence, StreetName and StreetNumber.

The last left join is used for [joinedCrashPeople] on ReportNumber and an

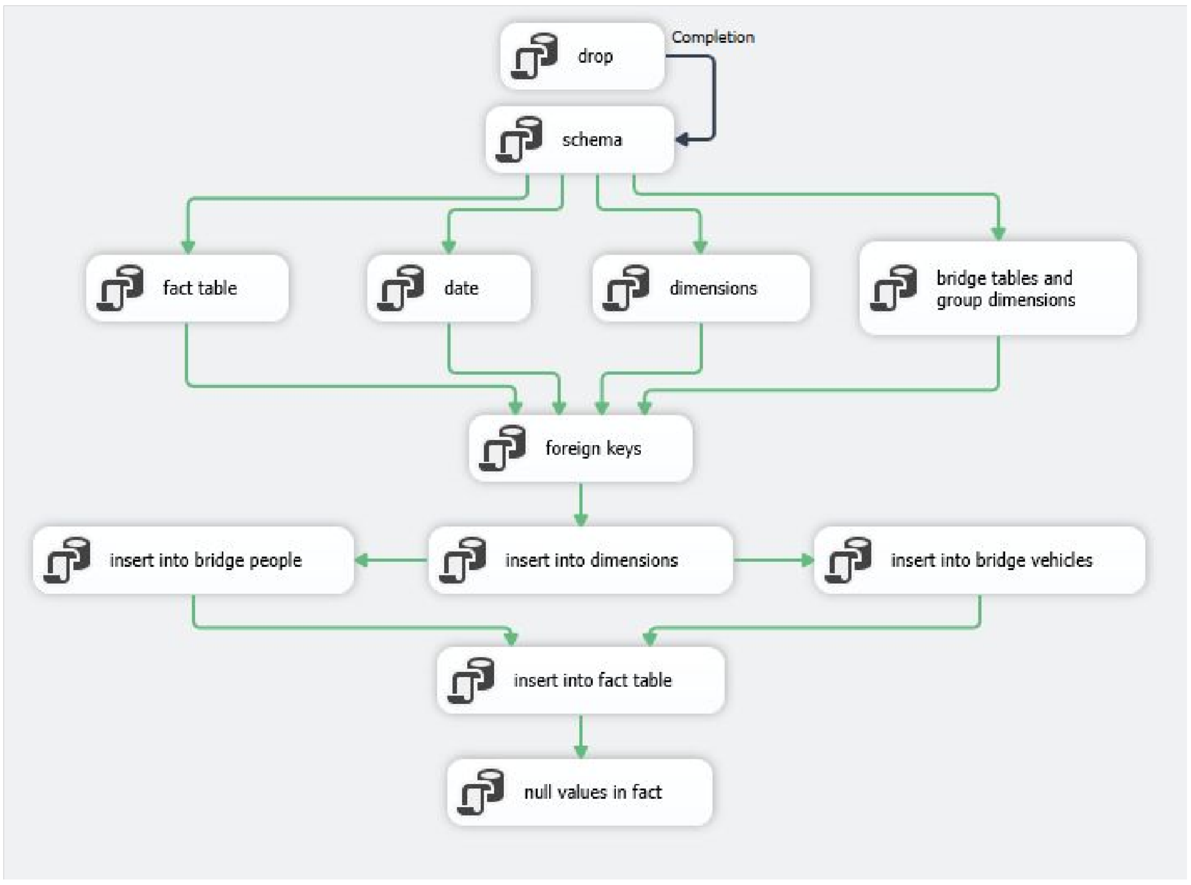
order by ReportNumber has been used which perfectly gives us the desired result.

v. Define proper relations between dimensions and fact tables

Facts table consists of foreign keys to following dimensions(which reference analogically their primary key):

- Location (LocationId)
- Environment (EnvironmentNo)
- Date (DateId)
- GroupVehicles (VehiclesGroup) -GroupPeople (PeoplesGroup).

It is related with above dimensions in a many-to-1 relationships. A crash can have one location, one date and one environment, but each of these dimensions can be associated with different crashes. Many-to-many relationships are forbidden in the world of entity-relationship modelling. We are opting to present our data in the way that each crash can have several people/vehicles involved and many people/vehicles can be involved in the same crash. That is why we have constructed the following intersect tables (*GroupVehicles* , *GroupPeople*) as well as (*BridgePeople* , *BridgeVehicle*) These are the “bridge”, which allows people/vehicles to be grouped together, whilst maintaining each individual entity. This allows a many-to-many relationships between fact table and *VehicleStatus/PeopleStatus* dimensions. *GroupVehicles/GroupPeople* have a primary key, referenced by fact table and also by foreign keys of *BridgePeople* and *BridgeVehicle* . Both bridges have a foreign key referencing *VehicleStatus* and *PeopleStatus* primary keys. Other than visible dimensions, we have two degenerate dimensions that are stored within our fact table, *PrimaryCause* and *SecondaryCause* . These dimensions have a single attribute, they are directly related to an event that the fact table stores, the crash. It is not eligible to be stored in a separate dimension table. The relationship between them and facts table is 1-1.

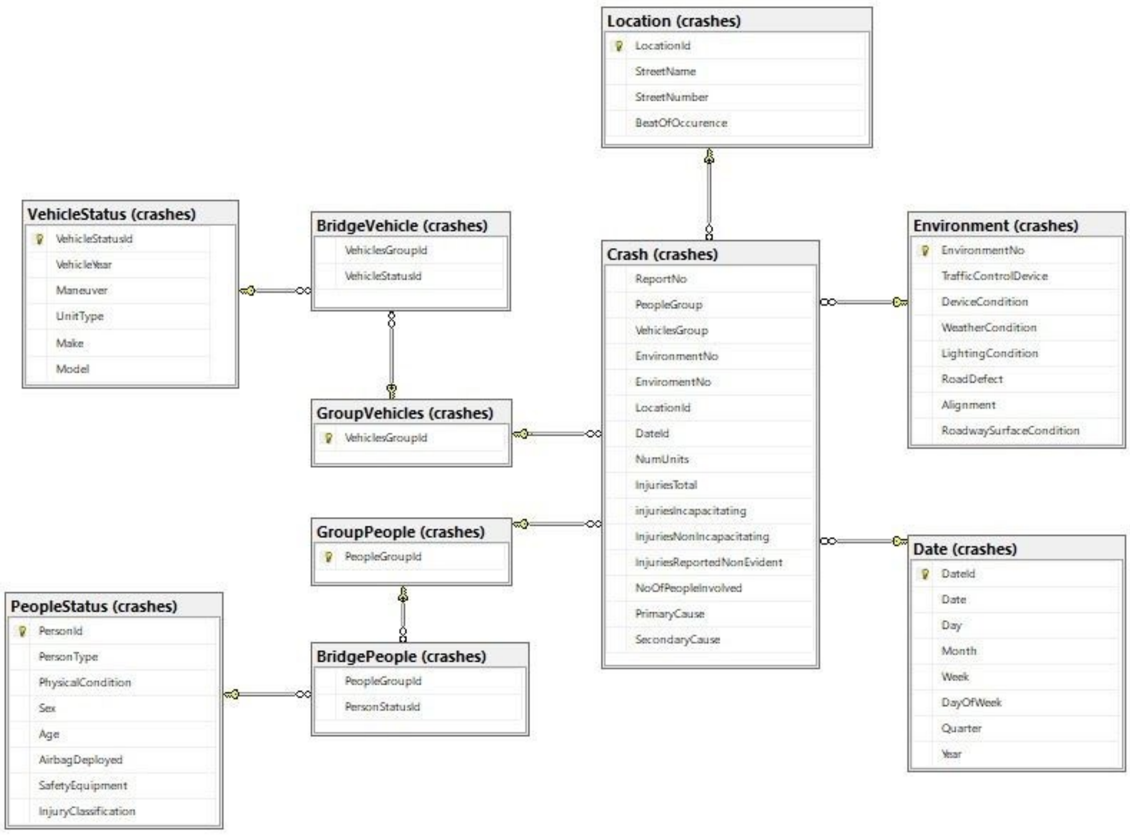


The picture above shows the SSIS package with the SQL instructions used to create all our multi-dimensional scheme.

- The schema is created first.
- Dimensions, bridges and the fact table are created in parallel once the schema is created.
- Foreign keys are defined all together after all tables are created.

- Data is inserted into tables starting from the dimensions *Location* , *Environment*, *Date*,*VehicleStatus* and *PeopleStatus* .
- *GroupPeople*, *GroupVehicles* , *BridgePeople* and *BridgeVehicle* are populated with data after inserting data into *VehicleStatus* and *PeopleStatus* , because they need those two dimensions.
- Finally, data is inserted into the fact table and some minor adjustments are made.

The following diagram represent our schema *crash* . The central fact table is connected directly to the dimensions *Location* , *Environment* and *Date* , and indirectly connected through bridge tables to the dimensions *VehicleStatus* and *PeopleStatus* .


Location (crashes)
LocationId
StreetName
StreetNumber
BeatOfOccurence
VehicleStatus (crashes)
VehicleStatusId
VehicleYear
Maneuver
UnitType
Make
Model
BridgeVehicle (crashes)
VehiclesGroupId
VehicleStatusId
Crash (crashes)
ReportNo
PeopleGroup
VehiclesGroup
EnvironmentNo
EnviromentNo
LocationId
DateId
NumUnits
InjuriesTotal
injuriesIncapacitating
InjuriesNonIncapacitating
InjuriesReportedNonEvident
NoOfPeopleInvolved
PrimaryCause
SecondaryCause
Environment (crashes)
EnvironmentNo
TrafficControlDevice
DeviceCondition
WeatherCondition
LightingCondition
RoadDefect
Alignment
RoadwaySurfaceCondition
GroupVehicles (crashes)
VehiclesGroupId
GroupPeople (crashes)
PeopleGroupId
Date (crashes)
DateId
Date
Day
Month
Week
DayOfWeek
Quarter
Year
PeopleStatus (crashes)
PersonId
PersonType
PhysicalCondition
Sex
Age
AirbagDeployed
SafetyEquipment
InjuryClassification
BridgePeople (crashes)
PeopleGroupId
PersonStatusId