Relational Data

301113 Programming for Data Science

WESTERN SYDNEY
UNIVERSITY

School of Computer, Data and Mathematical Sciences

Week 10 - Wickham and Grolemund, Ch 13

Outline

Motivation

You have been asked to analyse customer data to identify different purchasing patterns. The customer personal details are stored in one table but the purchased items are stored in another table. A third table contains the details of each product.

We need to be able to combine the tables to examine the relationships.

Outline

Examining Customer Purchases

We have worked with data from a single table, where the rows represent observations and the columns represent variables.

When working with databases, it is common to find that data is split across multiple tables, where each table is focused on a certain topic.

For example, a customer purchase database might contain the tables:

- Customer table {customer id, customer name, customer address}
- Product table {product id, product name, product description, product price}
- Purchase table {date of purchase, customer id, product id}

Examining Customer Purchases

We have worked with data from a single table, where the rows represent observations and the columns represent variables.

When working with databases, it is common to find that data is split across multiple tables, where each table is focused on a certain topic.

For example, a customer purchase database might contain the tables:

- Customer table {customer id, customer name, customer address}
- Product table {product id, product name, product description, product price}
- Purchase table {date of purchase, customer id, product id}

If we want to examine if there is an association between the types of products (product description) and the customer suburb (customer address), we need to merge all three tables into one. We can only merge the tables due to the relationships that exist between the tables.

What is relational data?

WESTERN SYDNEY UNIVERSITY

Relational Data

A set of tables is relational data if they contain variables that form relationships between the variables.

We often find that data is stored in relational tables in databases to reduce data redundancy and improve data integrity. For more information on database Normal forms `https://en.wikipedia.org/wiki/Database_normalization`.

When working with databases, large data, we are usually faced with relational data and so it is important that we know how to manage it for analysis.

Working with relational data

Most of the processing required to deal with relational tables is the joining of the tables. Once the tables are merged into one table, we can analyse that table, just as we analyse any other table.

We will be examining:

- Mutating Joins: add variables to a table from another table.
- Filtering Joins: filter observations based on their occurrence in another table.
- Set operations: treating observations as elements of a set.

Example Relational Data

Let's look at a set of table containing relational data.

Load the following libraries to access the data.

```
library(tidyverse)
library(nycflights13)
```

The data frames are:

- airlines: each airline and its carrier code
- airports: each airport, its code, position and time zone
- planes: each plane, with details
- weather: weather at each airport
- flights: details of each flight

Mapping out the relational data

The data frame `flights` contains information about each flight, but we find that it does not contain any details about:

- the age of the aircraft, the number of seats
- the name and location of the origin and destination airports
- the airline name
- the weather at the time of departure and arrival

Mapping out the relational data

The data frame `flights` contains information about each flight, but we find that it does not contain any details about:

- the age of the aircraft, the number of seats
- the name and location of the origin and destination airports
- the airline name
- the weather at the time of departure and arrival

But it does contain enough information so we can find these details from the other data frames.

- We can use the airport FAA code to lookup airport details.
- We can use the aircraft tail number to lookup details of the plane.
- We can use the carrier code to obtain the airline name.
- We can use the date and the airport code to lookup the weather.

Mapping out the relational data

The data frame `flights` contains information about each flight, but we find that it does not contain any details about:

- the age of the aircraft, the number of seats
- the name and location of the origin and destination airports
- the airline name
- the weather at the time of departure and arrival

But it does contain enough information so we can find these details from the other data frames.

- We can use the airport FAA code to lookup airport details.
- We can use the aircraft tail number to lookup details of the plane.
- We can use the carrier code to obtain the airline name.
- We can use the date and the airport code to lookup the weather.

To locate details, the tables must be linked (joined) together.

Plane Manufacturer

WESTERN SYDNEY UNIVERSITY

Problem

Who is the manufacturer of the plane from the first flight in the table `flights`. Use the table `flights` and `planes` to find this.

Outline

Database keys

We found that the set of tables each provide pieces of information and when combined, we obtain more detailed information.

The variables that provide the links between the two tables are called keys.

It is important that the key values be unique for each record in the table. If the same key value exists for multiple records, then when we lookup the key value, we won't know which record to use.

Database keys

We found that the set of tables each provide pieces of information and when combined, we obtain more detailed information.

The variables that provide the links between the two tables are called keys.

It is important that the key values be unique for each record in the table. If the same key value exists for multiple records, then when we lookup the key value, we won't know which record to use.

Problem

Identify the key variable in the `planes` table.

Types of keys

WESTERN SYDNEY UNIVERSITY

A variable can be one of two types of key:

- Primary Key: a unique identifier for an observation in a table. Primary keys are usually values such as serial numbers of ID numbers (such as student numbers). Given a value for a primary, we can find the single observation that matches it.
- Foreign key: a unique identifier for an observation in another table. Foreign keys are used to identify relationships to other tables.

Note that multiple variables can be used together as the primary key. For example, the `weather` table provides the weather at each airport for every hour, so the primary key is the `origin` and time variables combined.

Types of keys

A variable can be one of two types of key:

- Primary Key: a unique identifier for an observation in a table. Primary keys are usually values such as serial numbers of ID numbers (such as student numbers). Given a value for a primary, we can find the single observation that matches it.
- Foreign key: a unique identifier for an observation in another table. Foreign keys are used to identify relationships to other tables.

Note that multiple variables can be used together as the primary key. For example, the `weather` table provides the weather at each airport for every hour, so the primary key is the `origin` and time variables combined.

Problem

Which of the variables in the `flights` table are foreign keys?

Surrogate Keys

Keys are useful for combining tables. Keys are also useful for keeping track of observations before and after processing data.

Some tables don't contain primary keys.

We are able to add our own keys to data, for the purpose of keeping track of variables.

Surrogate Key

A surrogate key is a key that we have added to a table for the purpose of keeping track of each observation.
A simple way of adding a surrogate key is to create a variable containing the row number of the observation in the original table.

Outline

Joining Tables

The `flight` table contains the carrier ID, but not the name. To determine the name, we could:

1. Look up the carrier ID for the flight in the flight table
2. Find the row containing the same flight ID in the airlines table.
3. Read the carrier name from the found row of the airlines table.

Rather than looking up multiple tables, it is common practice to create a new temporary table, where the two (or more) tables are joined into one.

Once the tables are joined, all of the information will be in the same row (for the flight and the carrier name).

Joining Tables

WESTERN SYDNEY UNIVERSITY

The `flight` table contains the carrier ID, but not the name. To determine the name, we could:

1. Look up the carrier ID for the flight in the flight table
2. Find the row containing the same flight ID in the airlines table.
3. Read the carrier name from the found row of the airlines table.

Rather than looking up multiple tables, it is common practice to create a new temporary table, where the two (or more) tables are joined into one.

Once the tables are joined, all of the information will be in the same row (for the flight and the carrier name).

Example

Let's combine the flights and airlines tables

```
flights %>% select(year, month, day, hour, tailnum, carrier) %>%
    left_join(airlines, by = "carrier")
```

Types of Joins

If all primary keys are matched to foreign keys, the join provides a row for each match.

We get problems when:

- the foreign key variable does not contain all of the primary keys.
- the primary key variable does not contain all of the foreign keys.

Note that either or both of these can occur. When they do, we must make a decision of whether to leave out the observation, or include the observation with missing values.

Our choices are to use:

- inner join
- left outer join
- right outer join
- full outer join

Example Data

To demonstrate the result from performing each type of join, we will use the following small data frames.

```
# List of students and class they are enrolled in
students <- data.frame(
    name = c("Alice", "Bob", "Charlie", "Doris"),
    enrolledIn = c(1,2,5,6))

# List of class IDs and details of each class
classes <- data.frame(
    classId = c(1,2,3,4),
    className = c("Maths", "English", "Science", "History"))
```

Inner Join

An inner join creates a new table containing the rows from each table. The rows from each table are joined where the keys match. If there is no match, then the record is not included in the result.

```
students %>% inner_join(classes, by = c("enrolledIn" = "classId"))
```

```
   name enrolledIn className
1 Alice          1     Maths
2   Bob          2   English
```

For this case,

- "Charlie" and "Doris" from the left table are not included, and
- "Science" and "English" from the right table are not included.

Since they don't contain a match from the other table.

Left Outer Join

A left outer join creates a new table containing all rows from the left table and only rows from the right table that are matched to a row from the left table. NAs are inserted into all missing values.

```
students %>% left_join(classes, by = c("enrolledIn" = "classId"))
```

```
     name enrolledIn className
1   Alice          1     Maths
2     Bob          2   English
3 Charlie          5      <NA>
4   Doris          6      <NA>
```

For this case,

- "Charlie" and "Doris" from the left table are included with NA as the class name, and
- "Science" and "English" from the right table are not included

since they don't contain a match from the other table.

Right Outer Join

A right outer join creates a new table containing all rows from the right table and only rows from the left table that are matched to a row from the right table. NAs are inserted into all missing values.

```
students %>% right_join(classes, by = c("enrolledIn" = "classId"))
```

```
   name enrolledIn className
1 Alice          1     Maths
2   Bob          2   English
3  <NA>          3   Science
4  <NA>          4   History
```

For this case,

- “Charlie” and “Doris” from the left table are not included, and
- “Science” and “English” from the right table are included with NA as the student name

since they don’t contain a match from the other table.

Full Outer Join

WESTERN SYDNEY UNIVERSITY

A full outer join creates a new table containing all rows from the left and right tables. NAs are inserted into all missing values.

```
students %>% full_join(classes, by = c("enrolledIn" = "classId"))
```

```
     name enrolledIn className
1   Alice          1     Maths
2     Bob          2   English
3 Charlie          5      <NA>
4   Doris          6      <NA>
5    <NA>          3   Science
6    <NA>          4   History
```

For this case,

- “Charlie” and “Doris” from the left table are included with NA as the class name, and
- “Science” and “English” from the right table are included with NA as the student name

since they don’t contain a match from the other table.

Match the joins

Problem

Given the two tables `food` and `foodType`, being joined on the typeId:

```
       foodName typeId
1     Spaghetti      2
2     Chocolate      7
3 Chicken Salad      3
```

```
  typeId typeName
1      1      Pie
2      2    Pasta
3      3    Salad
```

Provide the tables created by performing an inner join and a left join.

Duplicate Keys

We often find that many records will reference the same item from another table.

```
# List of students and class they are enrolled in
students <- data.frame(
    name = c("Alice", "Bob", "Charlie", "Doris"),
    enrolledIn = c(1,2,2,6))
```

In this case, the rows matching the duplicate keys will be replicated.

```
students %>% left_join(classes, by = c("enrolledIn" = "classId"))
```

```
     name enrolledIn className
1   Alice          1     Maths
2     Bob          2   English
3 Charlie          2   English
4   Doris          6      <NA>
```

Duplicate keys in both tables

If there are duplicate keys in both the left and right tables, then there is likely an error in the data (since one table should contain a primary key). If both tables contain duplicate keys, then each row referenced by the duplicate is replicated.

```
classes <- data.frame(
    classId = c(1,1,2,3),
    className = c("Maths", "English", "Science", "History"))

students %>% left_join(classes, by = c("enrolledIn" = "classId"))
```

```
     name enrolledIn className
1   Alice          1     Maths
2   Alice          1   English
3     Bob          2   Science
4 Charlie          2   Science
5   Doris          6      <NA>
```

We find that "Alice" is repeated due to `classId` 1 appearing twice in the right table, and "Science" has appeared twice due to being references twice from the left table.

Filtering Joins

The joins we have examined combine the variables from two tables for a given set of observations.

Filtering joins filter a table based on the appearance of a key in another table.

Two types of filtering joins:

- Semi-join
- Anti-join

We will use this data to demonstrate these functions again.

```
# List of students and class they are enrolled in
students <- data.frame(
    name = c("Alice", "Bob", "Charlie", "Doris"),
    enrolledIn = c(1,2,5,6))

# List of class IDs and details of each class
classes <- data.frame(
    classId = c(1,2,3,4),
    className = c("Maths", "English", "Science", "History"))
```

Semi-join

A semi-join filters the observations from a table based on the appearance of a key in another table.

```
students %>% semi_join(classes, by = c("enrolledIn" = "classId"))
```

```
   name enrolledIn
1 Alice          1
2   Bob          2
```

Anti-join

An anti-join filters the observations from a table based on the absence of a key in another table.

```
students %>% anti_join(classes, by = c("enrolledIn" = "classId"))
```

```
     name enrolledIn
1 Charlie          5
2   Doris          6
```

Outline

SQL

SQL (Structured Query Language) is a database language that is used to manipulate and access content from many relational database systems.

This language was developed at IBM in the 1970 (originally called SEQUEL, Structured English Query Language). The language was further developed by Oracle in the early 1980s. The ANSI and ISO standards groups later used SQL as the standard database language in the late 1980s.

There are many databases programs that use SQL, but not all follow the standard. So there are often small differences in each.

SQL Statements

Each SQL statement is used to interact with the database. The most commonly used statements begin with one of the following verbs.

- `SELECT`: get data from a table
- `INSERT`: insert data into a table
- `UPDATE`: update existing tables
- `DELETE`: remove records from a table

A data scientist is likely to use the `SELECT` statement to retrieve data for analysis.

SQL Databases

An SQL database is a set of tables that usually have some relationship. The list of databases is stored in a table and can be viewed using

```
SHOW DATABASES;
```

To use a database:

```
USE databasename;
```

After selecting a database, we can then view the set of tables associated to the database.

```
SHOW TABLES;
```

SQL Database Tables

Each SQL database table has a name and a number of defined variables (columns), where each variable has a fixed type (e.g. character, binary, numeric, date).

A primary key might exist in the table.

To view the set of column names and their types, we must view the table schema

```
DESCRIBE tablename;
```

Extracting data from tables

To extract data from a database, we select the columns from the table.

```
# extract one column
SELECT column1 FROM tablename;
# extract multiple columns
SELECT column1, column2 FROM tablename;
# extract all columns
SELECT * FROM tablename;
```

Extracting data from tables

WESTERN SYDNEY UNIVERSITY

To extract data from a database, we select the columns from the table.

```
# extract one column
SELECT column1 FROM tablename;
# extract multiple columns
SELECT column1, column2 FROM tablename;
# extract all columns
SELECT * FROM tablename;
```

A set of conditions can be placed on the rows that are extracted.

```
SELECT column1 FROM tablename WHERE column2="text";
```

E.g.

```
SELECT cookingTime FROM recipe WHERE type = "Pie";
```

```
SELECT studentId FROM students WHERE age >= 20;
```

Maths students

WESTERN SYDNEY UNIVERSITY

Problem

Write an SQL statement using the `student` table below to provide the names of all students in a maths class.

```
     name   class
1   Alice   Maths
2     Bob English
3 Charlie Science
4   Doris History
```

Multiple Conditions

Conditions can be combined using the keywords AND, OR, NOT. Parentheses can be used to group complicated statements.

E.g.

```
SELECT studentId FROM students WHERE degree="Data Science" AND age >= 20;

SELECT country FROM countryInfo WHERE landMass > 50 AND gdp > 200;

SELECT machineID FROM network WHERE traffic > 40 AND NOT type = "router";

SELECT name FROM citizens
    WHERE country = "Australia" AND (city = "Melbourne" OR city = "Sydney);
```

Ordering Results

The results from the select statement can be ordered by any column and by multiple columns in the order that is provided. The order can be either ascending or descending.

```
SELECT studentId FROM students
    WHERE degree=”Data Science” AND age >= 20 ORDER BY date ASC;

SELECT country FROM countryInfo
    WHERE landMass > 50 AND gdp > 200, ORDER BY population;

SELECT machineID FROM network
    WHERE traffic > 40 AND NOT type = ”router” ORDER BY traffic;

SELECT name FROM citizens
    WHERE country = ”Australia” AND (city = ”Melbourne” OR city = ”Sydney)
    ORDER BY dob ASC, city DESC;
```

Joining Tables

WESTERN SYDNEY UNIVERSITY

SQL provides the same join operations that we used in R:

- JOIN: inner join
- LEFT JOIN: left outer join
- RIGHT JOIN: right outer join
- FULL JOIN: full outer join

Joins are performed with SELECT statements, so we need to be able to identify columns from multiple tables. SQL provides the `table.column` syntax.

Example SQL Joins

Joins can be performed with and without conditions. Note that SQL statements can be complicated, so they are usually stored in script files. Comments start with --.

```
-- join student and degree tables
SELECT students.name, degree.name
    FROM students INNER JOIN degree
    ON students.degreeId = degree.id;
```

Example SQL Joins

Joins can be performed with and without conditions. Note that SQL statements can be complicated, so they are usually stored in script files. Comments start with --.

```sql
-- join student and degree tables
SELECT students.name, degree.name
    FROM students INNER JOIN degree
    ON students.degreeId = degree.id;

-- only provide students with GPA > 4
SELECT students.name, degree.name
    FROM students LEFT JOIN degree
    ON students.degreeId = degree.id
    WHERE student.gpa > 4;
```

Example SQL Joins

Joins can be performed with and without conditions. Note that SQL statements can be complicated, so they are usually stored in script files. Comments start with --.

```
-- join student and degree tables
SELECT students.name, degree.name
    FROM students INNER JOIN degree
    ON students.degreeId = degree.id;

-- only provide students with GPA > 4
SELECT students.name, degree.name
    FROM students LEFT JOIN degree
    ON students.degreeId = degree.id
    WHERE student.gpa > 4;

-- and School is "Mathematics", then sort by degree name
SELECT students.name AS studentName, degree.name AS degreeName
    FROM students FULL JOIN degree
    ON students.degreeId = degree.id
    WHERE student.gpa > 4 AND degree.school = "Mathematics"
    ORDER BY degreeName;
```

Student classes

Problem

Given the student and class tables below, write an SQL statement to provide one table containing all of the student names and name of the class they are enrolled in.

	name	enrolledIn
1	Alice	1
2	Bob	2
3	Charlie	5
4	Doris	6

	classId	className
1	1	Maths
2	2	English
3	3	Science
4	4	History

Using Database Results

If we connect to a database server and provide a query, the results will be printed to the screen in a not so useful format.

Rather than connecting manually, we can connect to databases using other programs that interface with the database.

Many Web applications connect to a database backend and provide dynamic Web pages containing the content of the database results.

We can access databases using R and the DBI packages. To use the package, it must be installed.

```
install.packages("DBI")
```

Connecting to a database using DBI

The DBI interface can connect to many different DBI databases. Here we connect to an sqlite flatfile database (stored on the local machine). This database contains the details of school seminars.

```
library(DBI)
# establish a connection to the database
con <- dbConnect(RSQLite::SQLite(), dbname = "data/seminar.db")

# list the tables in the database
dbListTables(con)
```

```
[1] "maillist"      "seminar"       "seminar_state" "speaker"
[5] "users"         "venue"
```

```
# destroy the connection to the database
dbDisconnect(con)
```

Listing the table column names

The function `dbListFields` is used to provide the names of the given table from the provided database connection.

```
con <- dbConnect(RSQLite::SQLite(), dbname = "data/seminar.db")

# list the column names of the table "seminar"
dbListFields(con, "seminar")
```

```
[1] "id"          "state"      "title"      "abstract"   "time"
[6] "finish"      "speakerIds" "venueId"    "maillistId"
```

```
# list the column names of the table "seminar"
dbListFields(con, "venue")
```

```
[1] "id"   "name"
```

Now that we have the names of the columns, we can issue SQL queries.

Issuing queries

SQL queries can be issued to the database with the `dbGetQuery` function, using the database connection.

```
res <- dbGetQuery(con, "SELECT seminar.title, speaker.name
    FROM seminar LEFT JOIN speaker ON seminar.speakerIds = speaker.id
    WHERE seminar.id > 15 AND seminar.id < 20")
print(res)
```

```
                                            title             name
1   Open-world reasoning with unknown individuals Hector Levesque
2                  Revising with Several Formulas James Delgrande
3             Research topics in Wireless Networks     Ante Prodan
4 Why is a pure mathematician working in biology?  Andrew Francis
```

The results are provided as a data frame.

Seminar time and place

Problem

Given the column names from the previous slide, write the SQL statement that provides the time and venue name for seminars with ids 16, 17, 18 and 19. Order the results by their seminar id.

Closing the database connection

When finished accessing the database, the connection should be closed to free up memory and network sockets.

```r
# destroy the connection to the database
dbDisconnect(con)
```

If a connection is made many times using the same variable (e.g. `con`) R will eventually show a warning that it is closing the unused connections that are not associated to any variable.

Good programming practice

Whatever you open should eventually be closed.

Summary

- Relational data consists of a set of tables that contain variables providing the relationships between the tables.
- Key variables are used to identify related records across tables.
- Inner or outer joins are used to create temporary tables containing the information from multiple tables.
- SQL is a language used for accessing many relational databases.
- R provides an interface to SQL databases.