Introduction to Relational Databases

Principles of Data Science with R

Dr. Uma Ravat

PSTAT 10

Congratulations! We completed two modules!!

We completed

An Introduction to R programming for Data science

An Introduction to Probability for Data science

We also saw the important connection between the two that bridges **theory and applications.**

Next we start the last Module 3

Introductory Databases in R

We will learn work to with SQLite database software and use RSQLite package to work with databases through R.

Next we will see. . .

- Intro to Databases
- Connecting to database from R
  - the R packages `RSQLite`, `DBI`
  - the database on disk/file `Chinook_Sqlite.sqlite`
- SQL (Structured Query Language)
- 3 parts of the relational data model

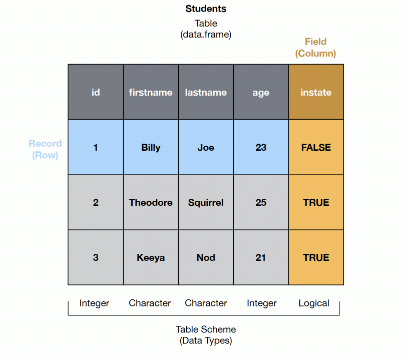
What is a Database?

Structured collection of data organized with

- efficient storage
- easy retrieval
- consistent management

dataframes in R to Tables in databases

R jargon	Database jargon	definition
column/variable	field/attribute	a variable/quantity of interest
row/sample/obs	record/tuple	collection of fields/attributes
dataframe	table/relation	a collection of records which all have the same fields/attribute (with different values)
types of the columns	table schema	datatype and other specifics about each field/attribute.

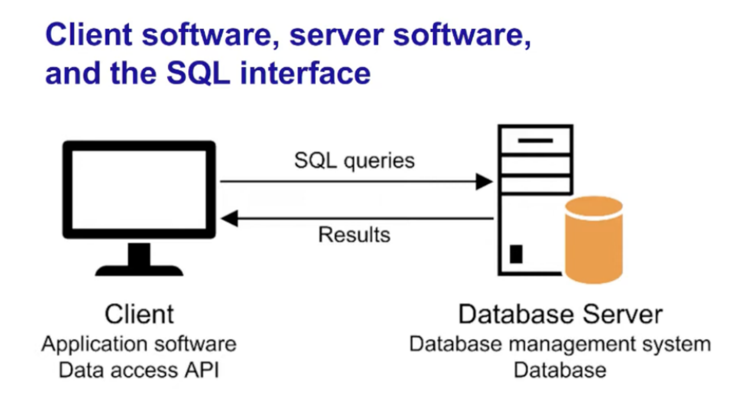


id	firstname	lastname	age	instate
1	Billy	Joe	23	FALSE
2	Theodore	Squirrel	25	TRUE
3	Keeya	Nod	21	TRUE
Integer	Character	Character	Integer	Logical

Why do we need database software?

- **Size**
  - R keeps its data frames in memory
  - Industrial databases are much bigger, need to store out of memory and bring into memory only required subsets
  - Must work with selected subsets
- **Speed**
  - Relational database model published in 70's by E. F. Codd at IBM labs in San Jose, CA.
  - Smart people have worked very hard making relational databases efficient
  - 2014 Turing award winner Michael Stonebraker
- **Concurrency**
  - Many users access the same database simultaneously
  - Potential for trouble (two users want to change the same record at once)
  - Database software takes care of this issue.

Client server architechure



Client software, server software, and the SQL interface

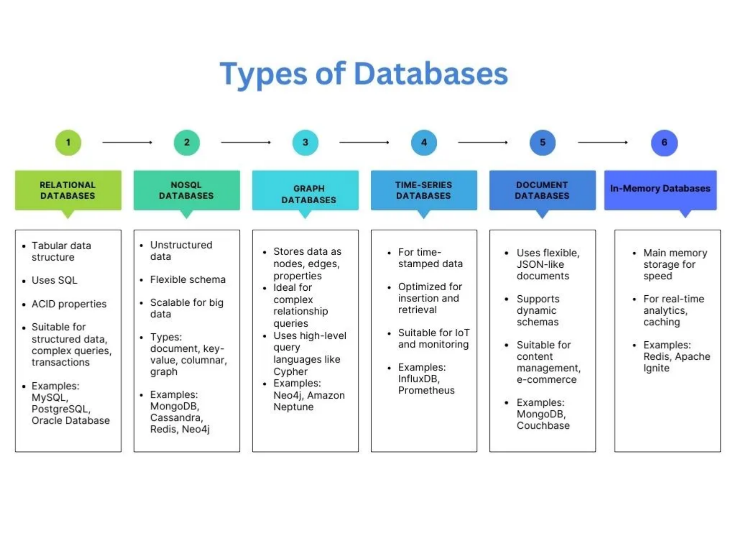
SQL queries

Results

Client
Application software
Data access API

Database Server
Database management system
Database

Database Manageement System (DBMS) Software



Types of Databases

RELATIONAL DATABASES

- Tabular data structure
- Uses SQL
- ACID properties
- Suitable for structured data, complex queries, transactions
- Examples: MySQL, PostgreSQL, Oracle Database

NOSQL DATABASES

- Unstructured data
- Flexible schema
- Scalable for big data
- Types: document, key-value, columnar, graph
- Examples: MongoDB, Cassandra, Redis, Neo4j

GRAPH DATABASES

- Stores data as nodes, edges, properties
- Ideal for complex relationship queries
- Uses high-level query languages like Cypher
- Examples: Neo4j, Amazon Neptune

TIME-SERIES DATABASES

- For time-stamped data
- Optimized for insertion and retrieval
- Suitable for IoT and monitoring
- Examples: InfluxDB, Prometheus

DOCUMENT DATABASES

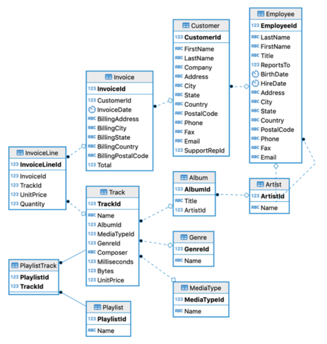
- Uses flexible, JSON-like documents
- Supports dynamic schemas
- Suitable for content management, e-commerce
- Examples: MongoDB, Couchbase

In-Memory Databases

- Main memory storage for speed
- For real-time analytics, caching
- Examples: Redis, Apache Ignite

What is a (relational) database?

- A **database** is a **collection of tables** that are **related** to one another, together **with specification of these relations.**



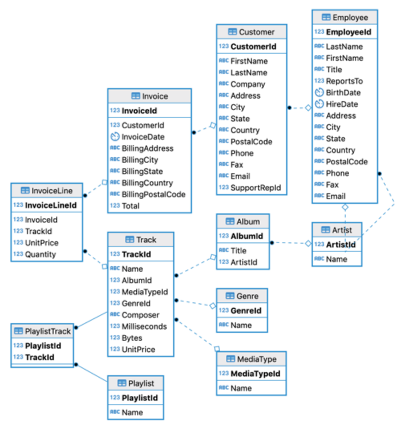
- An **entity relationship diagram** (ER diagram) helps visualize the structure of the tables and their relation to one another in a database.

Relational Database Management System (RDBMS)

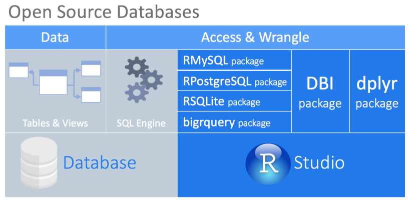
- data stored in tables that are related
- tables are linked/related to one another via keys
- **SQLite** is a popular light-weight, fast, full-featured RDBMS designed for simple applications (mobile apps)
    - is a simpler, file-based system that we will use via `RSQLite` package in `R`

Chinook DB

This DB represents a digital media store, including tables for artists, albums, media tracks, invoices, and customers.



Connecting R to SQLite



- First, we need to install the packages `DBI`, `RSQLite`,

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

- then we load them into our R session with `library()`

Also, we need a SQLite database file available at `Chinook_Sqlite.sqlite` for this lecture

Connecting to the database in R

```r
library(DBI)
library(RSQLite)
drv = dbDriver("SQLite")
chinook_db = dbConnect(drv,
                       dbname="./data/Chinook_Sqlite.sqlite")
```

The object `chinook_db` is now a persistent connection to the `Chinook_Sqlite.sqlite` database on disk.

Listing what's available

The data in a relational database is stored in relations, aka **tables**:

```
# List tables in our database
dbListTables(chinook_db)
```

```
##  [1] "Album"         "Artist"        "Customer"      "Employee"
##  [5] "Genre"         "Invoice"       "InvoiceLine"   "MediaType"
##  [9] "Playlist"      "PlaylistTrack" "Track"
```

Each table has rows of tuples, aka **records**, and columns of attributes, aka **fields**.

```
#List fields in Customer table
dbListFields(chinook_db, "Customer")
```

```
## [1] "CustomerId" "FirstName" "LastName" "Company" "Address"
## [6] "City" "State" "Country" "PostalCode" "Phone"
## [11] "Fax" "Email" "SupportRepId"
```

Importing a table as a data frame

```r
customer = dbReadTable(chinook_db, "Customer")
class(customer)
```

```
## [1] "data.frame"
```

```r
dim(customer)
```

```
## [1] 60 13
```

Now we could go on and perform R operations on `customer`, since it's a dataframe

Databases and SQL

- **SQL** (**structured query language**) is language/software to interact with data stored in databases
- **SQL** is the standard for database software
  - many different implementations of SQL, each with unique features.
- Most basic actions with databases are **SQL queries**, like row/column selections, inserts, updates and deletes

SELECT

Main tool in the SQL language: **SELECT**, which allows you to perform queries on a particular table in a database. It has the form:

```
SELECT columns
  FROM table
  WHERE condition
  GROUP BY columns
  HAVING condition
  ORDER BY column [ASC | DESC]
  LIMIT offset, count;
```

WHERE, GROUP BY, HAVING, ORDER BY, LIMIT are all optional

Example

Pick out five columns from the table “Customer”, and only look at the first 6 rows:

```
dbGetQuery(chinook_db,
           "select CustomerId, FirstName, LastName, City, Country
           from Customer
           limit 6" )
```

```
##   CustomerId FirstName    LastName                City        Country
## 1          1      Luís   Gonçalves São José dos Campos         Brazil
## 2          2    Leonie      Köhler           Stuttgart        Germany
## 3          3  François    Tremblay            Montréal         Canada
## 4          4     Bjørn      Hansen                Oslo         Norway
## 5          5 František Wichterlová              Prague Czech Republic
## 6          6    Helena        Holý              Prague Czech Republic
```

To replicate this simple command on the imported data frame:

```
customer[1:6, c("CustomerId", "FirstName", "LastName","City", "Country")]
```

```
##   CustomerId FirstName    LastName                City        Country
## 1          1      Luís   Gonçalves São José dos Campos         Brazil
## 2          2    Leonie      Köhler           Stuttgart        Germany
## 3          3  François    Tremblay            Montréal         Canada
## 4          4     Bjørn      Hansen                Oslo         Norway
## 5          5 František Wichterlová              Prague Czech Republic
## 6          6    Helena        Holý              Prague Czech Republic
```

ORDER BY

We can use the ORDER BY option in SELECT to specify an ordering for the rows

Default is ascending order; add DESC for descending

```
dbGetQuery(chinook_db,
           "select CustomerId, FirstName, LastName, City, Country
                        from Customer
                        ORDER BY FirstName DESC
                        limit 10")
```

```
##    CustomerId FirstName   LastName           City        Country
## 1          42     Wyatt     Girard       Bordeaux         France
## 2          25    Victor    Stevens        Madison            USA
## 3          19       Tim      Goyer      Cupertino            USA
## 4          44     Terhi Hämäläinen       Helsinki        Finland
## 5          54     Steve     Murray     Edinburgh  United Kingdom
## 6          49 Stanisław     Wójcik         Warsaw         Poland
## 7          12   Roberto    Almeida Rio de Janeiro         Brazil
## 8          29    Robert      Brown        Toronto         Canada
## 9          26   Richard Cunningham     Fort Worth            USA
## 10         59      Puja Srivastava      Bangalore          India
```

Close connection

We opened a connection as follows:

```
chinook_db <- dbConnect(SQLite(),
                  "Chinook_Sqlite.sqlite")
```

After the end of a session, it is good practice to explicitly close your connection.

```
dbDisconnect(chinook_db)
```

If indeed the connection is closed, reading some data should give an error

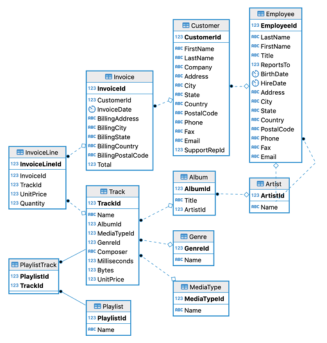
```
dbGetQuery(chinook_db, "select CustomerId, FirstName, LastName from Customer")
```

Error in h(simpleError(msg, call)) : error in evaluating the argument 'conn' in selecting a method for function 'dbGetQuery': object 'chinook_db' not found

Back to some theory: The relational model

In the **relational database**, the data is organized into **relations/tables.**

A relational model is used to represent data and the relationships between data items



The Relational Model consist of 3 parts

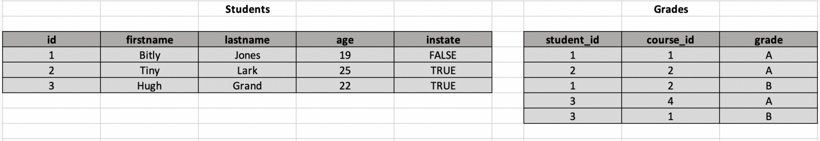
1. Manipulative part
    - Allows for data to be manipulated in the database.
    - SQL for create, update, delete tables, databases and user access
    - SQL for select, insert, update, delete data in tables
    - We saw some SQL queries for Select; more later
2. Structural part
    - Tables, relations between tables and rules/constraints for these
    - Visually as ER diagram: Table schema and relations between tables
    - Database Schema and rules/constraints, Primary Keys and Foreign Keys.
3. Integrity part
    - Rules to maintain integrity of data
    - Necessary to keep data complete, consistent and reliable.
        - Entity integrity: integrity of each relation
        - Referential integrity : integrity between relations

Structural part : Relations

Table/Relation

- Each relation must have a **unique name** in the database
- **No duplicate rows**, or tuples, are allowed in a table/relation
- **Each row** in a table/relation has **its own unique key**

Why Keys?

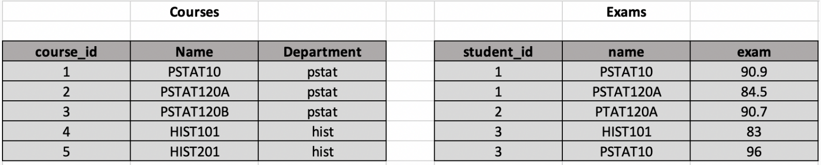


Students

id	firstname	lastname	age	instate
1	Bitly	Jones	19	FALSE
2	Tiny	Lark	25	TRUE
3	Hugh	Grand	22	TRUE

Grades

student_id	course_id	grade
1	1	A
2	2	A
1	2	B
3	4	A
3	1	B



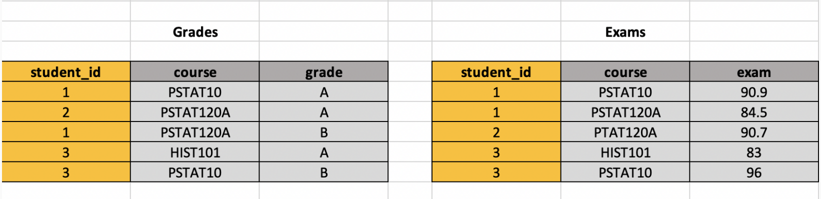
Courses

course_id	Name	Department
1	PSTAT10	pstat
2	PSTAT120A	pstat
3	PSTAT120B	pstat
4	HIST101	hist
5	HIST201	hist

Exams

student_id	name	exam
1	PSTAT10	90.9
1	PSTAT120A	84.5
2	PTAT120A	90.7
3	HIST101	83
3	PSTAT10	96

Why Keys?



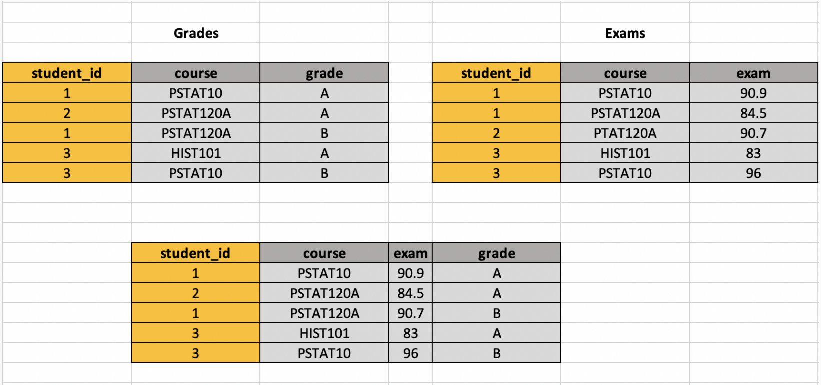
Grades

student_id	course	grade
1	PSTAT10	A
2	PSTAT120A	A
1	PSTAT120A	B
3	HIST101	A
3	PSTAT10	B

Exams

student_id	course	exam
1	PSTAT10	90.9
1	PSTAT120A	84.5
2	PTAT120A	90.7
3	HIST101	83
3	PSTAT10	96

Why Keys?



Grades

student_id	course	grade
1	PSTAT10	A
2	PSTAT120A	A
1	PSTAT120A	B
3	HIST101	A
3	PSTAT10	B

Exams

student_id	course	exam
1	PSTAT10	90.9
1	PSTAT120A	84.5
2	PTAT120A	90.7
3	HIST101	83
3	PSTAT10	96

student_id	course	exam	grade
1	PSTAT10	90.9	A
2	PSTAT120A	84.5	A
1	PSTAT120A	90.7	B
3	HIST101	83	A
3	PSTAT10	96	B

Structural part : Keys

A Key

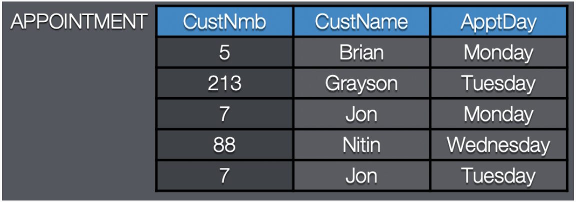
- consists of one or more attributes
- places certain constraints on a databases
- Each row in a table/relation has its own unique key (primary key)
- used to establish and identify relationships between relations
  - a row in a relation can be linked with another row in other relations (foreign key)

4 types of keys

- Super key
- Candidate key
- Primary key
- Foreign key

Keys: Super Keys

A super key is a set of one or more attributes that uniquely identify a tuple in a relation.



APPOINTMENT

CustNmb	CustName	ApptDay
5	Brian	Monday
213	Grayson	Tuesday
7	Jon	Monday
88	Nitin	Wednesday
7	Jon	Tuesday

Super Keys for the Appointment relation:

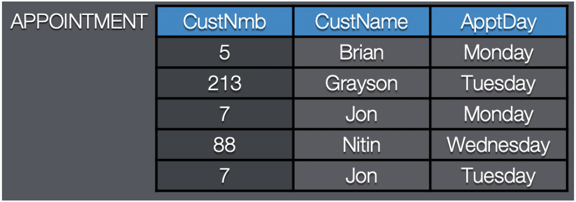
- {CustNmb, CustName, ApptDay},
- {CustNmb, ApptDay},
- {CustName, ApptDay}

What about {CustNmb, CustName}, {CustNmb},{CustName}, {ApptDay} ?

Keys: Candidate Keys

- Candidate key is a minimal set of attributes which can uniquely identify a tuple.
- Candidate keys are selected from the set of super keys.
- Candidate keys should not have any redundant attributes.
- There can be more than one candidate key in a relation
- The candidate key can be simple, having only one attribute, or it can be a composite of multiple attributes.
- A candidate key is a super key, but not the other way round

Keys: Candidate Keys



APPOINTMENT

CustNmb	CustName	ApptDay
5	Brian	Monday
213	Grayson	Tuesday
7	Jon	Monday
88	Nitin	Wednesday
7	Jon	Tuesday

Candidate Keys for the Appointment relation

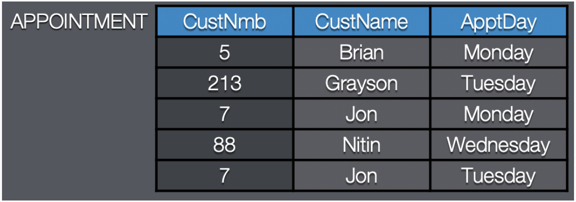
- {CustNmb, ApptDay},
- {CustName, ApptDay}

What about {CustNmb, CustName, ApptDay}, {CustNmb, CustName}, {CustNmb},{CustName}, {ApptDay} ?

Keys: Primary Keys

- A primary key is the minimal set of attributes which can uniquely identify a tuple.
- one of the candidate keys is chosen as the primary key.
- Candidate keys NOT selected for use as the primary key are called Alternate Keys.
- The same primary key cannot be used for different relations
- Primary key values cannot be null

Keys: Primary Keys



APPOINTMENT

CustNmb	CustName	ApptDay
5	Brian	Monday
213	Grayson	Tuesday
7	Jon	Monday
88	Nitin	Wednesday
7	Jon	Tuesday

The primary key for our APPOINTMENT relation is

- {CustNmb, ApptDay}

What about {CustNmb, CustName, ApptDay}, {CustNmb, CustName}, {CustNmb},{CustName}, {ApptDay} ?

What about {CustName, ApptDay}?

Back to Chinook DB : Field metadata

Unlike a dataframe, there is extra information in a database table that expresses relational information between tables.

```
dbGetQuery(chinook_db, "pragma table_info(Customer)")
```

```
##    cid         name         type notnull dflt_value pk
## 1    0   CustomerId      INTEGER       1         NA  1
## 2    1    FirstName NVARCHAR(40)       1         NA  0
## 3    2     LastName NVARCHAR(20)       1         NA  0
## 4    3      Company NVARCHAR(80)       0         NA  0
## 5    4      Address NVARCHAR(70)       0         NA  0
## 6    5         City NVARCHAR(40)       0         NA  0
## 7    6        State NVARCHAR(40)       0         NA  0
## 8    7      Country NVARCHAR(40)       0         NA  0
## 9    8   PostalCode NVARCHAR(10)       0         NA  0
## 10   9        Phone NVARCHAR(24)       0         NA  0
## 11  10          Fax NVARCHAR(24)       0         NA  0
## 12  11        Email NVARCHAR(60)       1         NA  0
## 13  12 SupportRepId      INTEGER       0         NA  0
```

KEYS: Primary key

The **primary key** is a *unique identifier* of the rows in a table. Two rows cannot have the same primary key:

```
dbGetQuery(chinook_db,
    "select CustomerId, FirstName, LastName, City, Country
        from Customer
        limit 2")
```

```
##   CustomerId FirstName  LastName                City Country
## 1          1      Luís Gonçalves São José dos Campos  Brazil
## 2          2    Leonie    Köhler           Stuttgart Germany
```

```r
dbExecute(chinook_db,
    paste("insert into customer",
        "(CustomerId, FirstName, LastName, Email)",
        "values",
        "(1, 'Luis','Armstrong','LuisArmstrong@pstat.ucsb.edu')"))
```

```
## Error: UNIQUE constraint failed: Customer.CustomerId
```

CustomerId is the **primary key** and must be unique.

Multi-column primary key

Primary key's can consist of multiple columns if it takes multiple columns to identify a row in a table. But, two rows cannot have the same primary key.

```
# Single column primary key
dbGetQuery(chinook_db, "pragma table_info(Customer)")
```

```
##    cid         name         type notnull dflt_value pk
## 1    0   CustomerId      INTEGER       1         NA  1
## 2    1    FirstName NVARCHAR(40)       1         NA  0
## 3    2     LastName NVARCHAR(20)       1         NA  0
## 4    3      Company NVARCHAR(80)       0         NA  0
## 5    4      Address NVARCHAR(70)       0         NA  0
## 6    5         City NVARCHAR(40)       0         NA  0
## 7    6        State NVARCHAR(40)       0         NA  0
## 8    7      Country NVARCHAR(40)       0         NA  0
## 9    8   PostalCode NVARCHAR(10)       0         NA  0
## 10   9        Phone NVARCHAR(24)       0         NA  0
## 11  10          Fax NVARCHAR(24)       0         NA  0
## 12  11        Email NVARCHAR(60)       1         NA  0
## 13  12 SupportRepId      INTEGER       0         NA  0
```

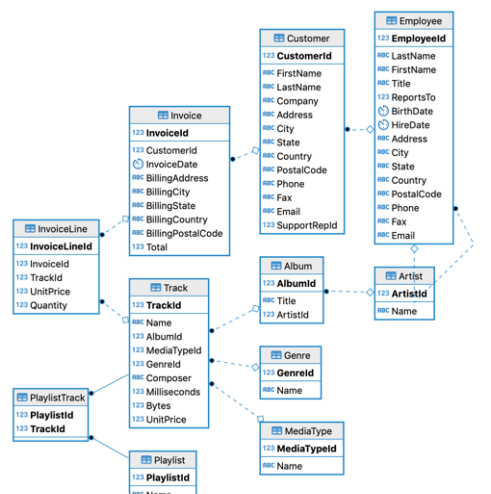
```
# Multi column primary key
dbGetQuery(chinook_db, "pragma table_info(PlayListTrack)")
```

```
##   cid       name    type notnull dflt_value pk
## 1   0 PlaylistId INTEGER       1         NA  1
## 2   1    TrackId INTEGER       1         NA  2
```

KEYS: Primary key

Tables are not required to have a primary key, but most do. All the tables in Chinook have a primary key.

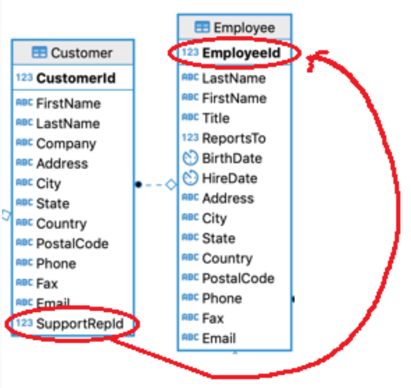
In ER diagram, Primary Key is denoted as **bold** or underlined field name.



KEYS: Foreign Keys

The foreign key is a set of attributes that matches the primary key for another relation. A foreign key field *points to* the primary key of another table.

The foreign key links two relations and can be used to cross-reference the relations.



Customer
- **CustomerId**
- FirstName
- LastName
- Company
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email
- SupportRepId

Employee
- **EmployeeId**
- LastName
- FirstName
- Title
- ReportsTo
- BirthDate
- HireDate
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email

KEYS: Foreign Keys

The relationship between tables is expressed by primary keys and **foreign keys**.

Remember we are working with a relational database, following a relational data model.

```
dbGetQuery(chinook_db,
           "pragma foreign_key_list(customer)")
```

```
##   id seq    table        from         to on_update on_delete match
## 1  0   0 Employee SupportRepId EmployeeId NO ACTION NO ACTION  NONE
```

KEYS: Enforcing Foreign keys

Foreign keys must either point to an existing value or be NULL.

To enforce Foreign key constraints in SQLite RDBMS

```
# Required for foreign-key support otherwise foreign keys are not enforced
dbExecute(chinook_db, "pragma foreign_keys = on")
```

```
dbGetQuery(chinook_db,
           "SELECT max(EmployeeId) FROM Employee")
```

```
##   max(EmployeeId)
## 1               8
```

```
dbGetQuery(chinook_db,
           "SELECT max(CustomerId) FROM Customer")
```

```
##   max(CustomerId)
## 1             888
```

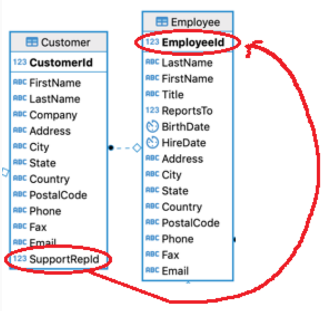
```r
dbExecute(chinook_db,
    "INSERT INTO Customer
    (CustomerId, FirstName, LastName, Email, SupportRepId)
    VALUES
    (59, 'Luis', 'Armstrong', 'luisArmstrong@pstat.ucsb.edu', 88)")
```

```
## Error: UNIQUE constraint failed: Customer.CustomerId
```

```r
dbExecute(chinook_db,
    "INSERT INTO Customer
    (CustomerId, FirstName, LastName, Email, SupportRepId)
    VALUES
    (60, 'Luis', 'Armstrong', 'luisArmstrong@pstat.ucsb.edu', 10)")
```

```
## Error: FOREIGN KEY constraint failed
```

Interpretation of foreign key



Customer
- CustomerId
- FirstName
- LastName
- Company
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email
- SupportRepId

Employee
- EmployeeId
- LastName
- FirstName
- Title
- ReportsTo
- BirthDate
- HireDate
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email

- Each customer in Customer table *can* be assigned a support representative
- The support rep is an employee at the store and therefore has a unique id, EmployeeId
- This unique id,EmployeeId , is the primary key of the employee table

Thus real-world relationship is encoded by the relational model using primary and foreign key relationships.

Keys: Why do we need keys?

- For identifying any row of data in a relation uniquely
- They ensure integrity of data is maintained.
- Keys establish relationships between relations and identify relationships between relations

The Relational Model consist of 3 parts

1. Manipulative part
   - Allows for data is manipulated in the database.
   - SQL for create, update, delete tables, databases and user access
   - SQL for select, insert, update, delete data in tables
   - We saw some SQL queries for Select; more later
2. Structural part
   - Tables, relations between tables and rules/constraints for these
   - Visually as ER diagram: Table schema and relations between tables
   - Database Schema and rules/constraints, Primary Keys and Foreign Keys.
3. Integrity part
   - Rules to maintain integrity of data
   - Necessary to keep data complete, consistent and reliable.
     - **Entity integrity:** integrity of each relation
     - **Referential integrity:** integrity between relations

Integrity Constraints

We have seen two examples of *integrity constraints*:

- **Entity integrity:** Primary keys must be unique (and not NULL)
    - This ensures there are no duplicate records
- **Referential integrity:** Foreign keys must reference existing primary keys or be NULL
    - ensures that cross-references to non-existing tuples cannot occur

These constraints enforce the *integrity* of a database; no bad data or corrupted relationships.

Keys help maintain the integrity of the data

Database Schema

The **schema** of a database describes its *structure*:

- Names of all the tables
- Names of all fields in each table
- Primary key/foreign key relationships between tables
- Other metadata (data types of each field in each table, . . .)

Basically everything other than the actual data itself.

Represented via E-R diagrams (Entity relationship)

We have been looking at parts of the schema with the `pragma` keyword.

```
dbGetQuery(chinook_db, "pragma table_info(customer)")
```

```
##    cid         name         type notnull dflt_value pk
## 1    0   CustomerId      INTEGER       1         NA  1
## 2    1    FirstName NVARCHAR(40)       1         NA  0
## 3    2     LastName NVARCHAR(20)       1         NA  0
## 4    3      Company NVARCHAR(80)       0         NA  0
## 5    4      Address NVARCHAR(70)       0         NA  0
```

We saw…

- Databases are used to store massive amounts of data that cannot fit in memory.
- SQL is the language used to manipulate relational databases
- SQLite is the SQL implementation we will use, provided by the `RSQLite` package.