Introduction to Structured Query Language (SQL)

Ying XU

Assistant Professor

Engineering Systems and Design (ESD)

Singapore University of Technology and Design

Edited based on Peter Jackson's slides on "Database Design and Joins" of DBA 2017

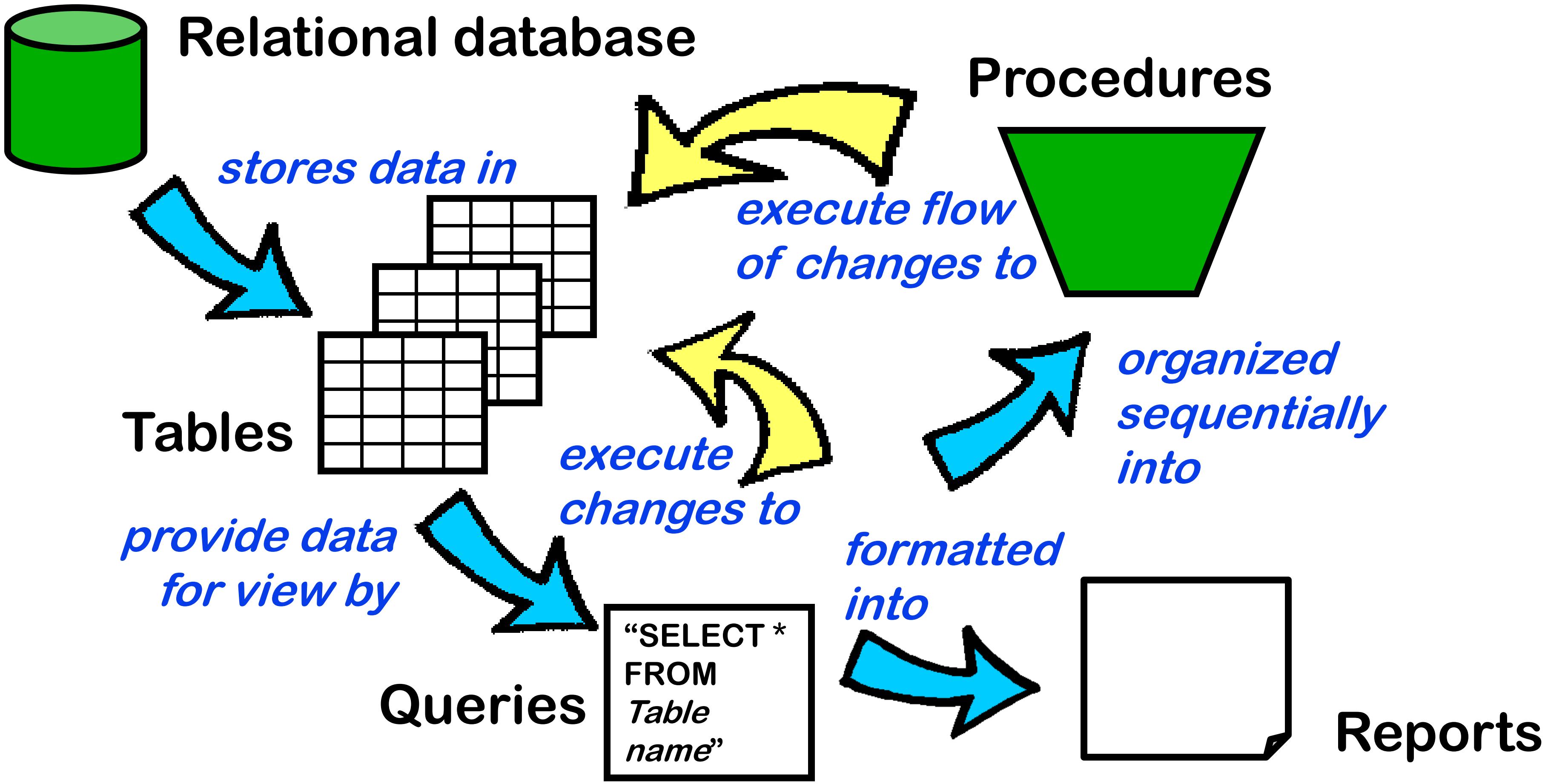
Introduction

- *Structured Query Language (SQL) let you access and manipulate databases*
- SQL can
  - execute queries against a database
  - retrieve data from a database
  - create views in a database
  - Insert/ update/ delete records in a database
  - create new databases
  - create new tables in a database
  - create stored procedures in a database
  - ……….

Origin of SQL (Structured Query Language)

- *1970: "A Relational Model of Data for Large Shared Data Banks" E.F. Codd*
  - *Definitive model for relational databases*
- *1970's: Chamberlin and Boyce at IBM publish SEQUEL language*
- *1978: IBM successfully tests prototype with customers*
- *1980's: commercial implementations of RDBMS (relational database management systems)*
- *1986: SQL adopted as standard by American National Standards Institute (ANSI)*

Relational Database Management System (RDBMS)



Relational database

stores data in

Tables

Procedures

execute flow of changes to

execute changes to

organized sequentially into

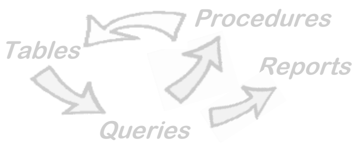
provide data for view by

Queries

"SELECT * FROM *Table name*"

formatted into

Reports



Overview

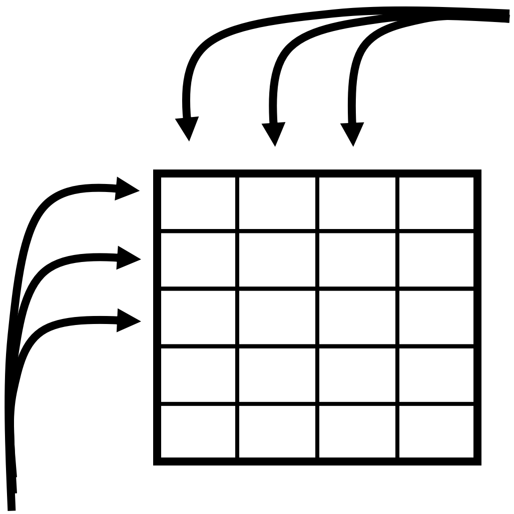
- Tables
  - Sample Table with Key
  - Use Keys to Make Relationships
  - Tables are Related by Keys
- Queries
  - Query Result
- Save the Query

- Selective Query Result
  - Select Fields You are Interested In
  - Select Records You are interested in
- Use Select as a Calculator

- Aggregate Data Query
  - Aggregate Query with Sub-Totals

- Make a New Table
- Modify Data in the Database

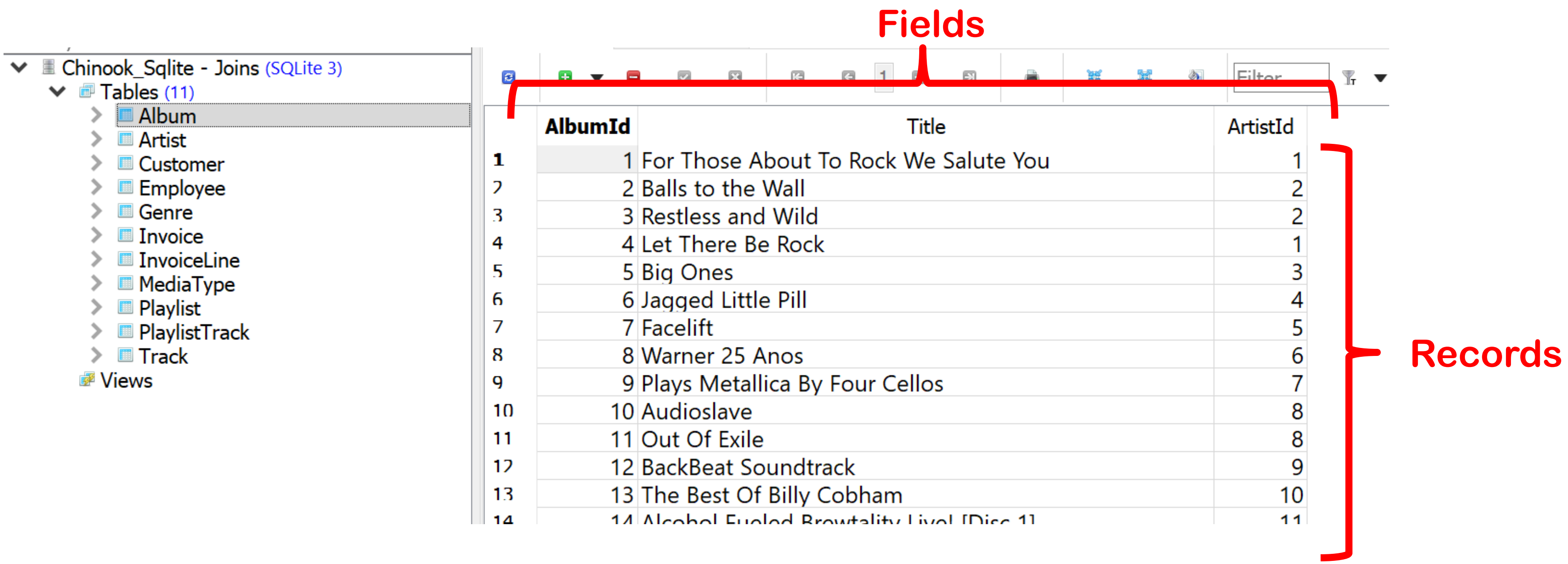
Tables



Fields: a fixed number of columns, each column having a prescribed data type (integer, single, double, text, …) and length

Records: an unlimited number of rows, each row containing data in each column of the prescribed type

Album Table



Fields

Records

	AlbumId	Title	ArtistId
1	1	For Those About To Rock We Salute You	1
2	2	Balls to the Wall	2
3	3	Restless and Wild	2
4	4	Let There Be Rock	1
5	5	Big Ones	3
6	6	Jagged Little Pill	4
7	7	Facelift	5
8	8	Warner 25 Anos	6
9	9	Plays Metallica By Four Cellos	7
10	10	Audioslave	8
11	11	Out Of Exile	8
12	12	BackBeat Soundtrack	9
13	13	The Best Of Billy Cobham	10
14	14	Alcohol Fueled Brewtality Live! [Disc 1]	11

Album Table Structure

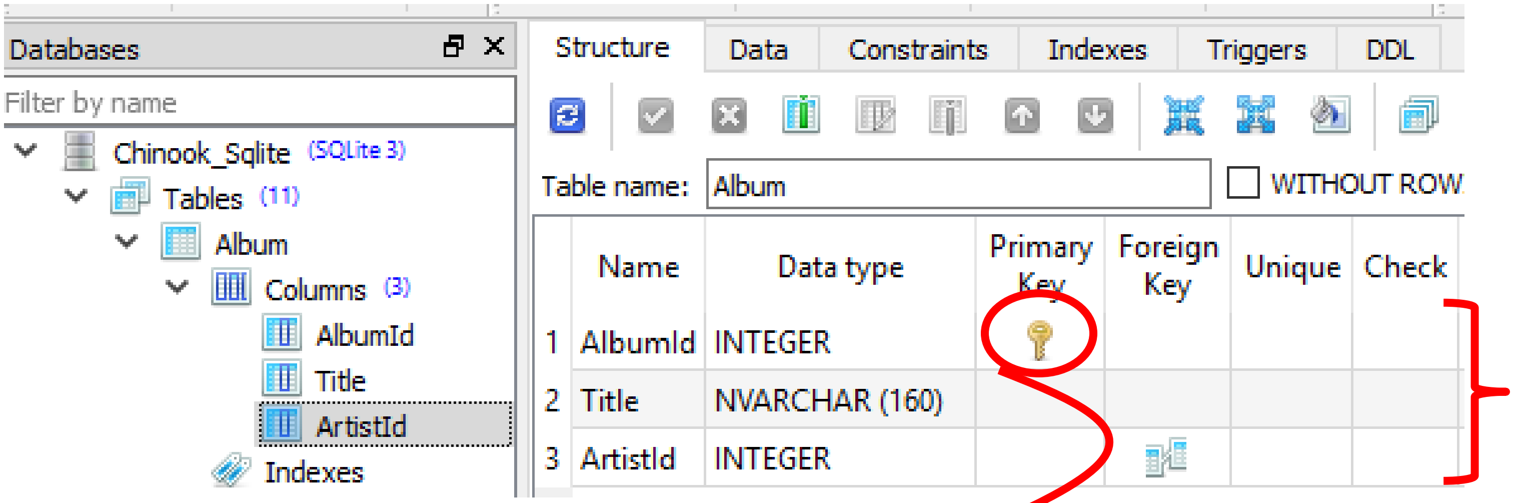


Table name: Album

□ WITHOUT ROW

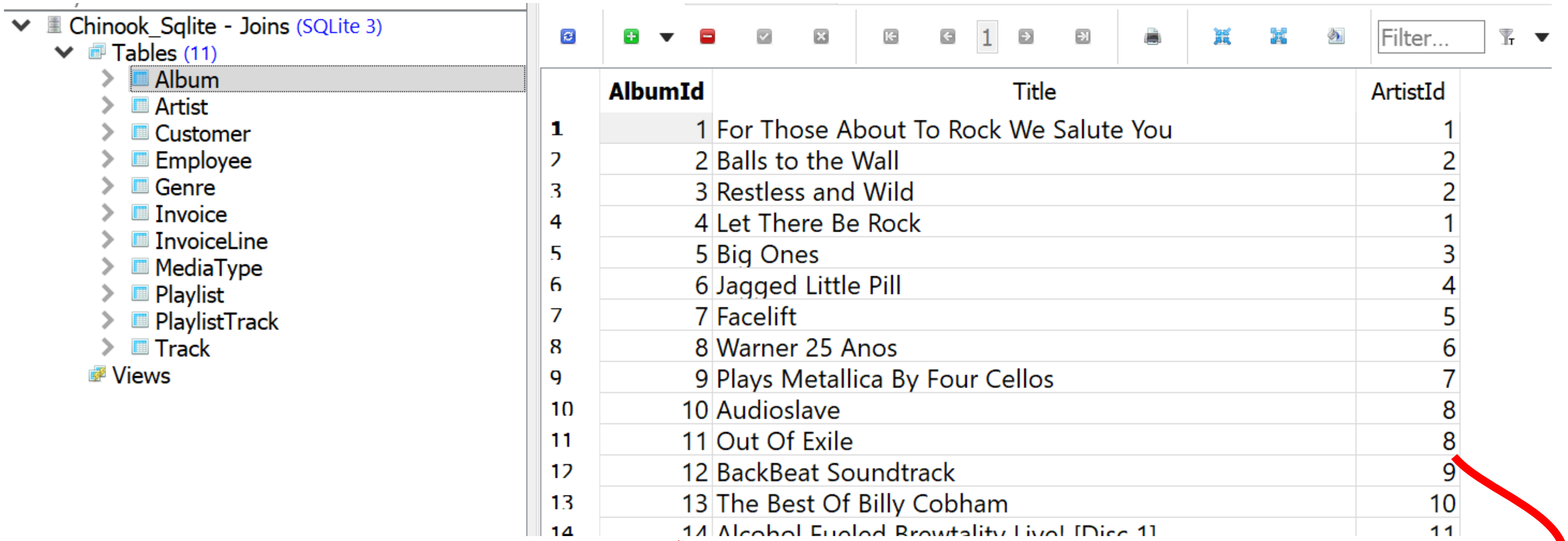
	Name	Data type	Primary Key	Foreign Key	Unique	Check
1	AlbumId	INTEGER	🔑			
2	Title	NVARCHAR (160)				
3	ArtistId	INTEGER		●		

Fields with data type and other properties

Primary Key: **a field, or collection of fields, whose data uniquely identify a record**

Notice: AlbumId is a PRIMARY KEY so it <u>must be</u> unique to each record

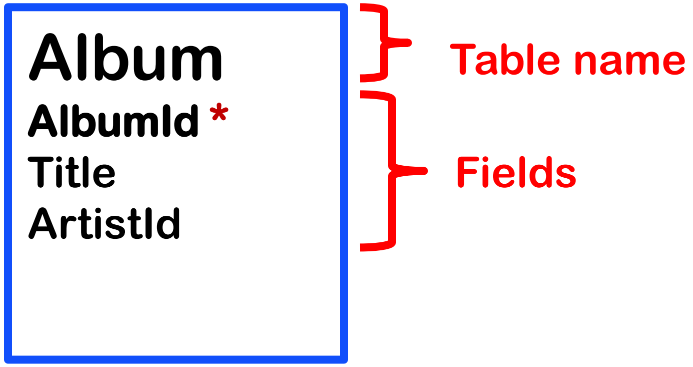
Album Table: Primary Key



	AlbumId	Title	ArtistId
1	1	For Those About To Rock We Salute You	1
2	2	Balls to the Wall	2
3	3	Restless and Wild	2
4	4	Let There Be Rock	1
5	5	Big Ones	3
6	6	Jagged Little Pill	4
7	7	Facelift	5
8	8	Warner 25 Anos	6
9	9	Plays Metallica By Four Cellos	7
10	10	Audioslave	8
11	11	Out Of Exile	8
12	12	BackBeat Soundtrack	9
13	13	The Best Of Billy Cobham	10
14	14	Alcohol Fueled Brewtality Live! [Disc 1]	11

Notice: AlbumId is unique to each record, ArtistId is not

Album Table Block Representation



Album

AlbumId *

Title

ArtistId

Table name

Fields

* Primary Key
(shown in bold)

Album Table: Foreign Key

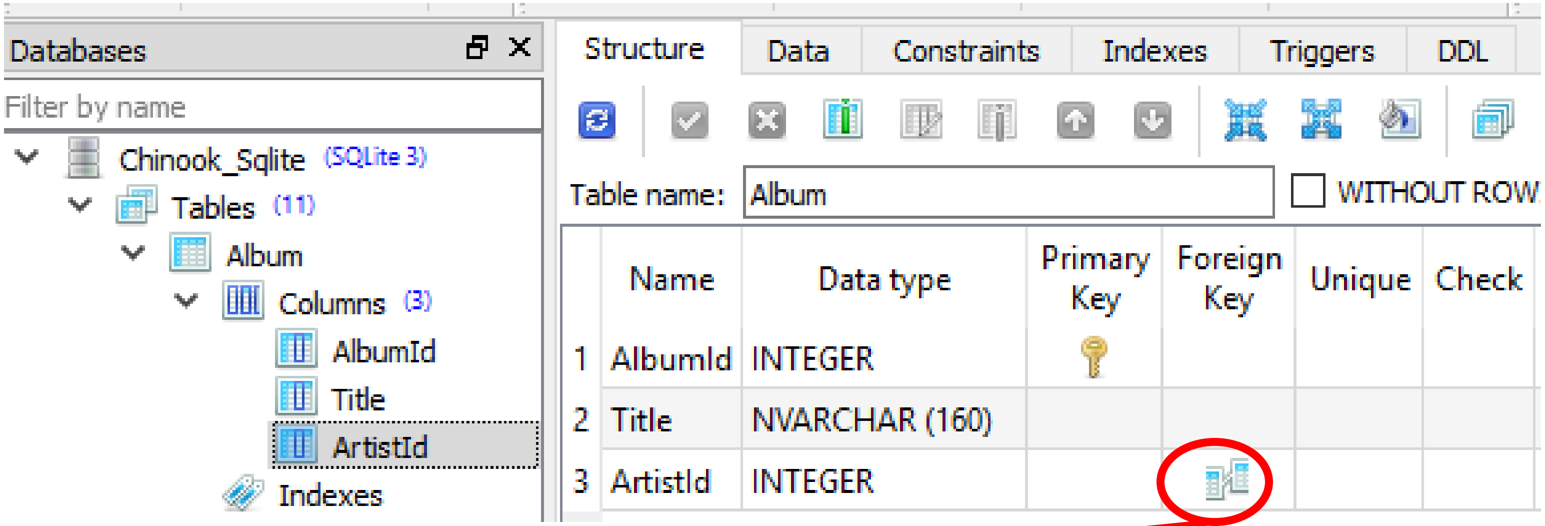


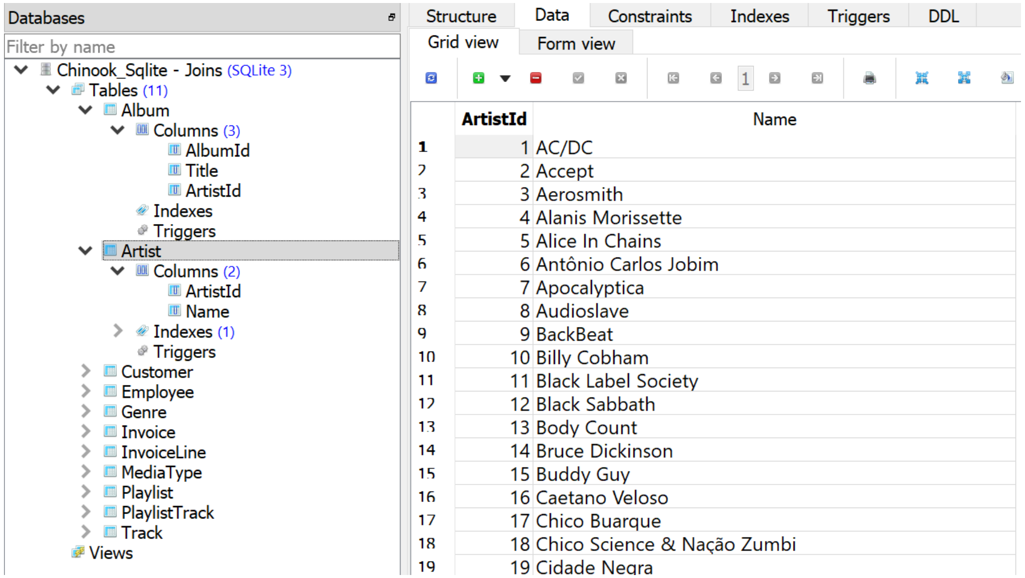
Table name: Album ☐ WITHOUT ROW

	Name	Data type	Primary Key	Foreign Key	Unique	Check
1	AlbumId	INTEGER	🔑			
2	Title	NVARCHAR (160)				
3	ArtistId	INTEGER		🔗		

Foreign Key: **a field, or collection of fields, in one table that refers to the PRIMARY KEY in another table**

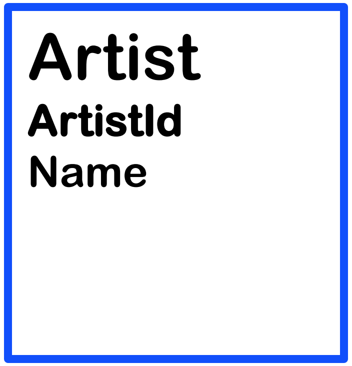
Notice: ArtistId is a FOREIGN KEY as it <u>is the PRIMARY KEY in the Artist Table</u>

The Artist Table Block Representation



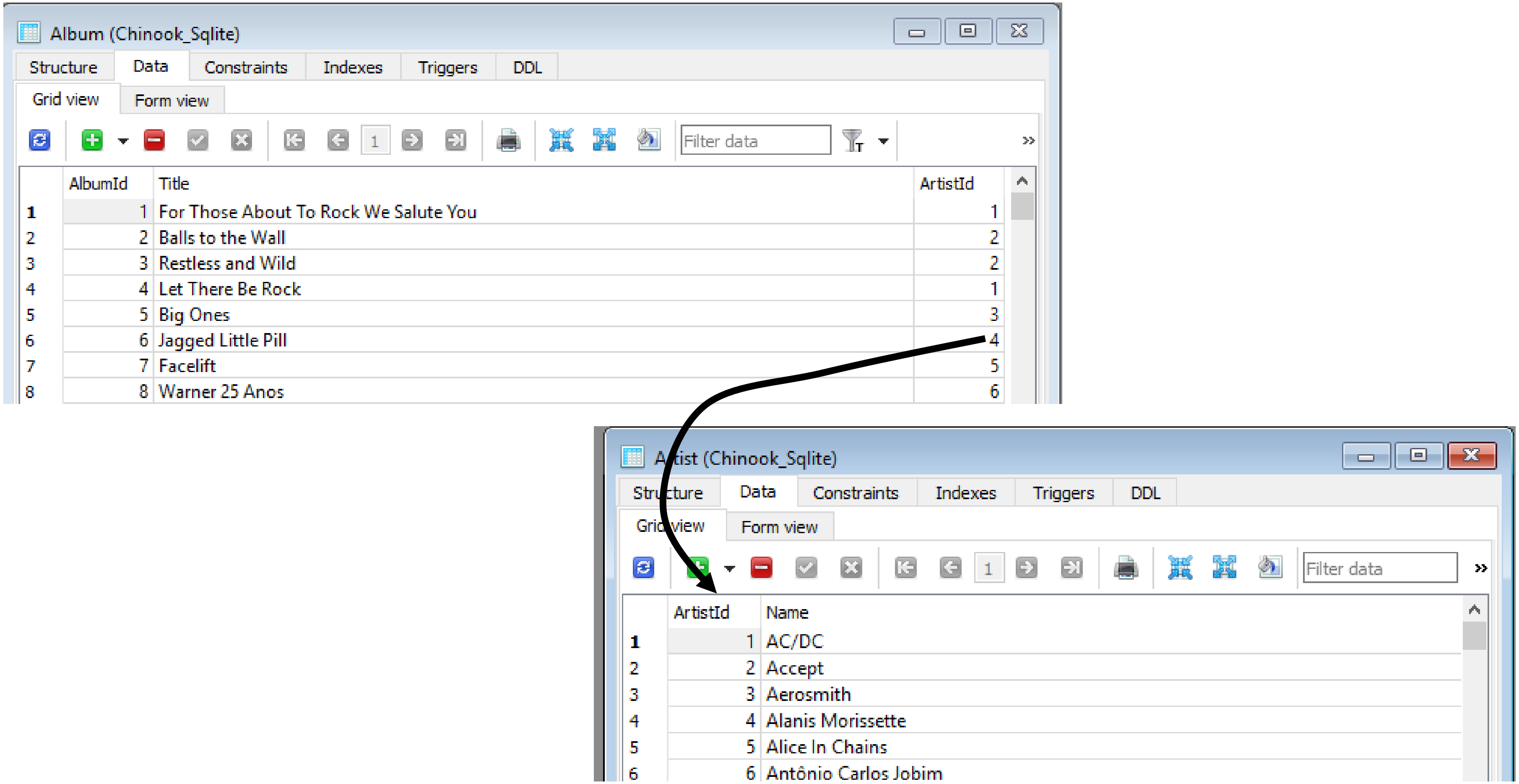
- Chinook_Sqlite - Joins (SQLite 3)
  - Tables (11)
    - Album
      - Columns (3)
        - AlbumId
        - Title
        - ArtistId
      - Indexes
      - Triggers
    - Artist
      - Columns (2)
        - ArtistId
        - Name
      - Indexes (1)
      - Triggers
    - Customer
    - Employee
    - Genre
    - Invoice
    - InvoiceLine
    - MediaType
    - Playlist
    - PlaylistTrack
    - Track
  - Views

	ArtistId	Name
1	1	AC/DC
2	2	Accept
3	3	Aerosmith
4	4	Alanis Morissette
5	5	Alice In Chains
6	6	Antônio Carlos Jobim
7	7	Apocalyptica
8	8	Audioslave
9	9	BackBeat
10	10	Billy Cobham
11	11	Black Label Society
12	12	Black Sabbath
13	13	Body Count
14	14	Bruce Dickinson
15	15	Buddy Guy
16	16	Caetano Veloso
17	17	Chico Buarque
18	18	Chico Science & Nação Zumbi
19	19	Cidade Negra


Artist
ArtistId
Name

ArtistId (shown in bold) is the unique (PRIMARY KEY) field for the Artist table

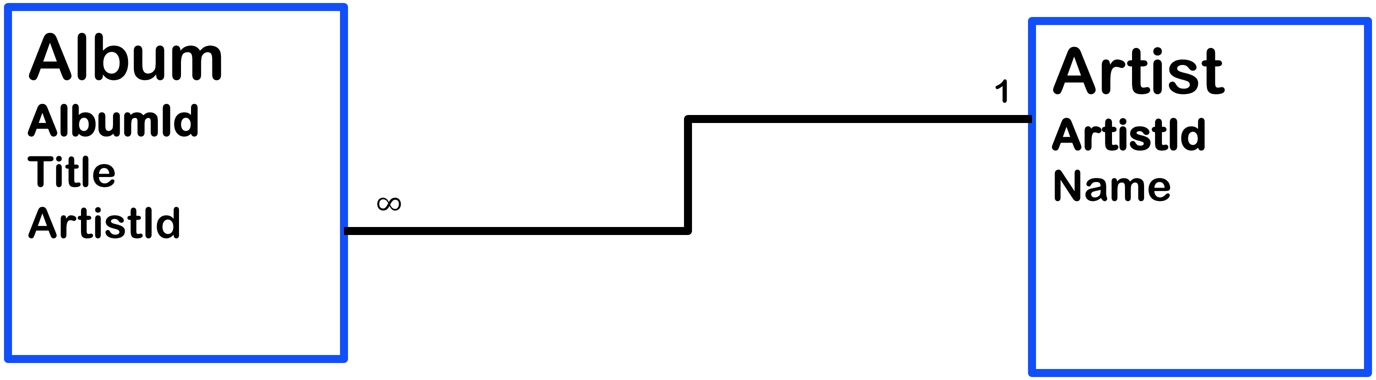
Use Keys to Make Relationships



	AlbumId	Title	ArtistId
1	1	For Those About To Rock We Salute You	1
2	2	Balls to the Wall	2
3	3	Restless and Wild	2
4	4	Let There Be Rock	1
5	5	Big Ones	3
6	6	Jagged Little Pill	4
7	7	Facelift	5
8	8	Warner 25 Anos	6

	ArtistId	Name
1	1	AC/DC
2	2	Accept
3	3	Aerosmith
4	4	Alanis Morissette
5	5	Alice In Chains
6	6	Antônio Carlos Jobim

One-to-Many Relationship



Album
AlbumId
Title
ArtistId

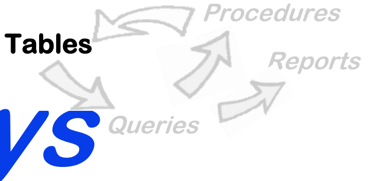
∞

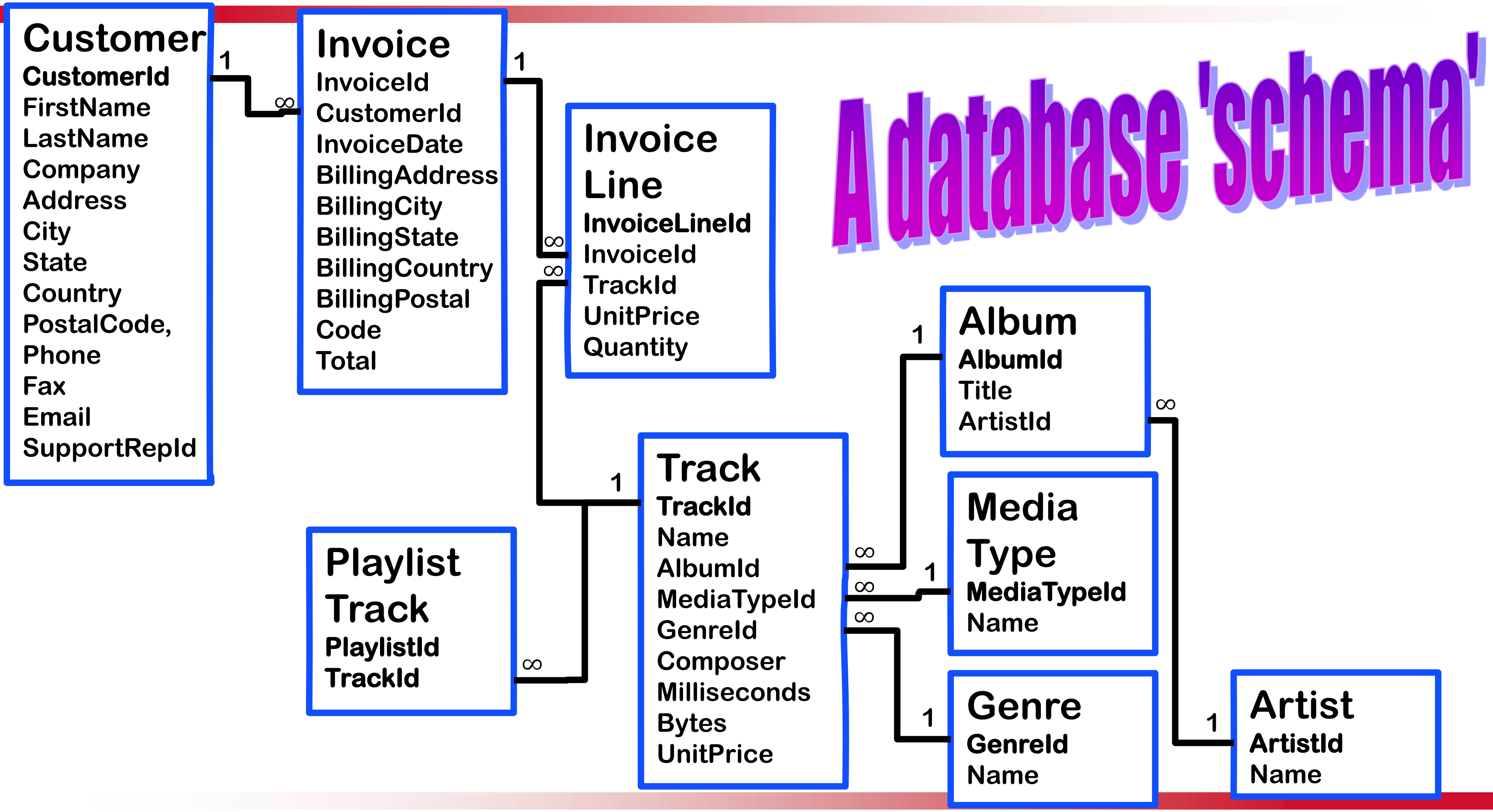
1

Artist
ArtistId
Name

A particular ArtistId will occur only once in table Artist, but could occur many (∞) times in table Album

Tables are Related by Keys





A database 'schema'

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

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostal
Code
Total

Invoice Line
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

Playlist Track
PlaylistId
TrackId

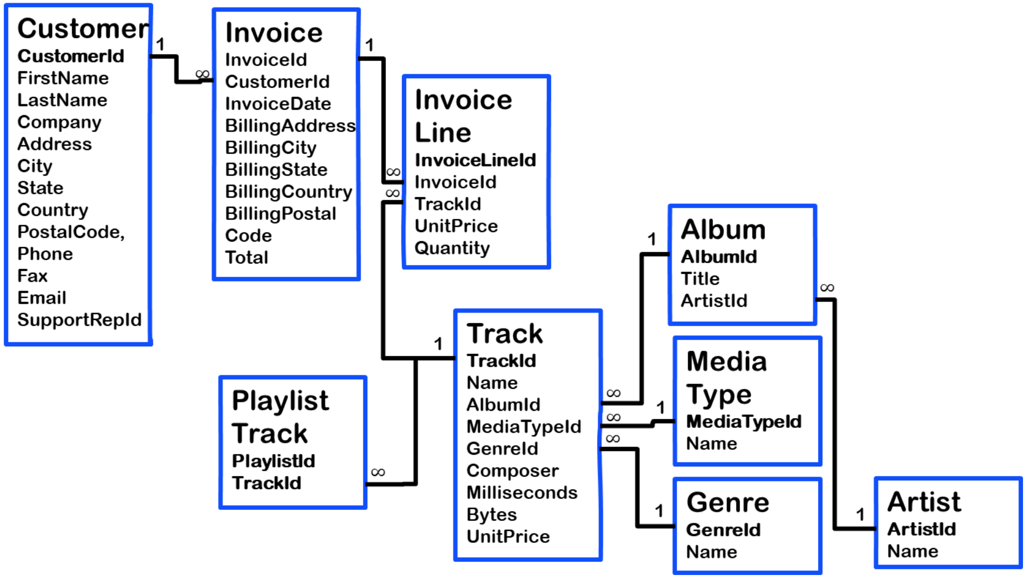
Media Type
MediaTypeId
Name

Genre
GenreId
Name

Artist
ArtistId
Name

Database Schema

- *Schema: map of tables, fields, and relationships in a relational database*
- *Schema = database design*



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

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostal
Code
Total

Invoice Line
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Album
AlbumId
Title
ArtistId

Playlist Track
PlaylistId
TrackId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

Media Type
MediaTypeId
Name

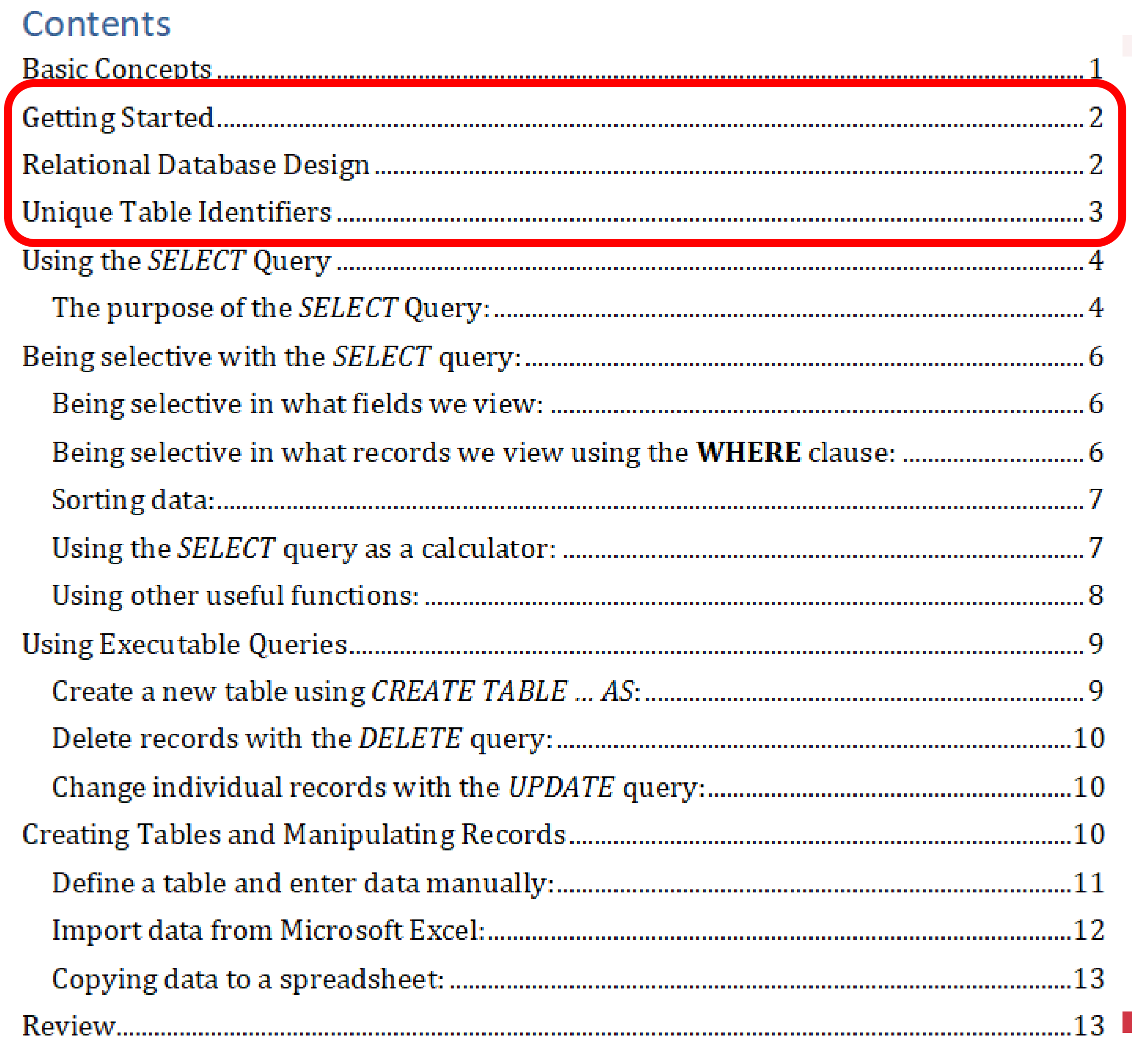
Genre
GenreId
Name

Artist
ArtistId
Name

Database Design Principle

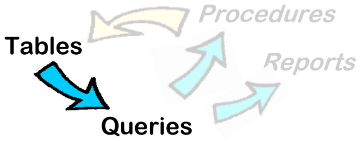
- Avoid duplicating data
  - Don’t put customer phone number into Orders table
    - Same phone number then occurs in hundreds or thousands of records
    - Wasted storage
    - Subject to error: person recording new order may make typing mistake and enter phone number incorrectly
  - Set up Customers table (one record per customer) and include phone number field

SQLite Activity



Contents

Queries



Query Example:

"SELECT * FROM Invoice"

English-like statement describing data request

Non-procedural: does not describe HOW to get data (looping, testing)

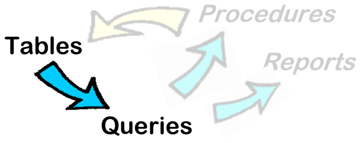
Prescribed syntax: "KEYWORD *expression* KEYWORD *expression* ..."

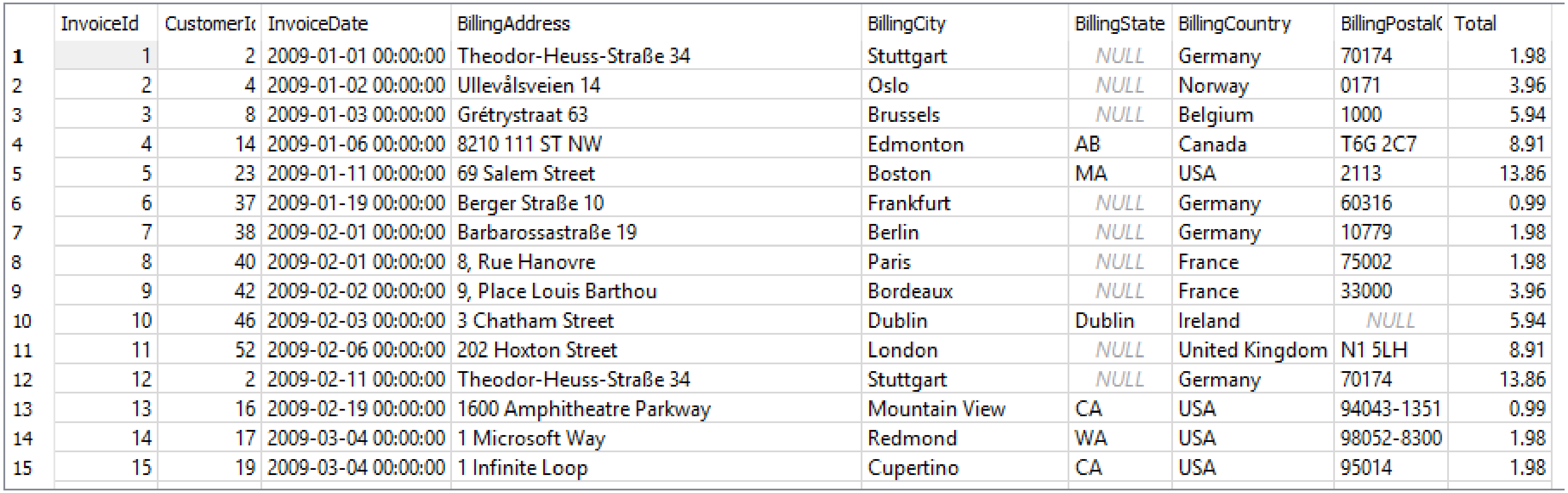
Small set of keywords: SELECT, FROM, AS, WHERE, ORDER BY, CASE WHEN…

Expression: *field_name (column_name), table_name, conditions,….*

Note: SQL keywords are NOT case sensitive: select is the same as SELECT

Query Result





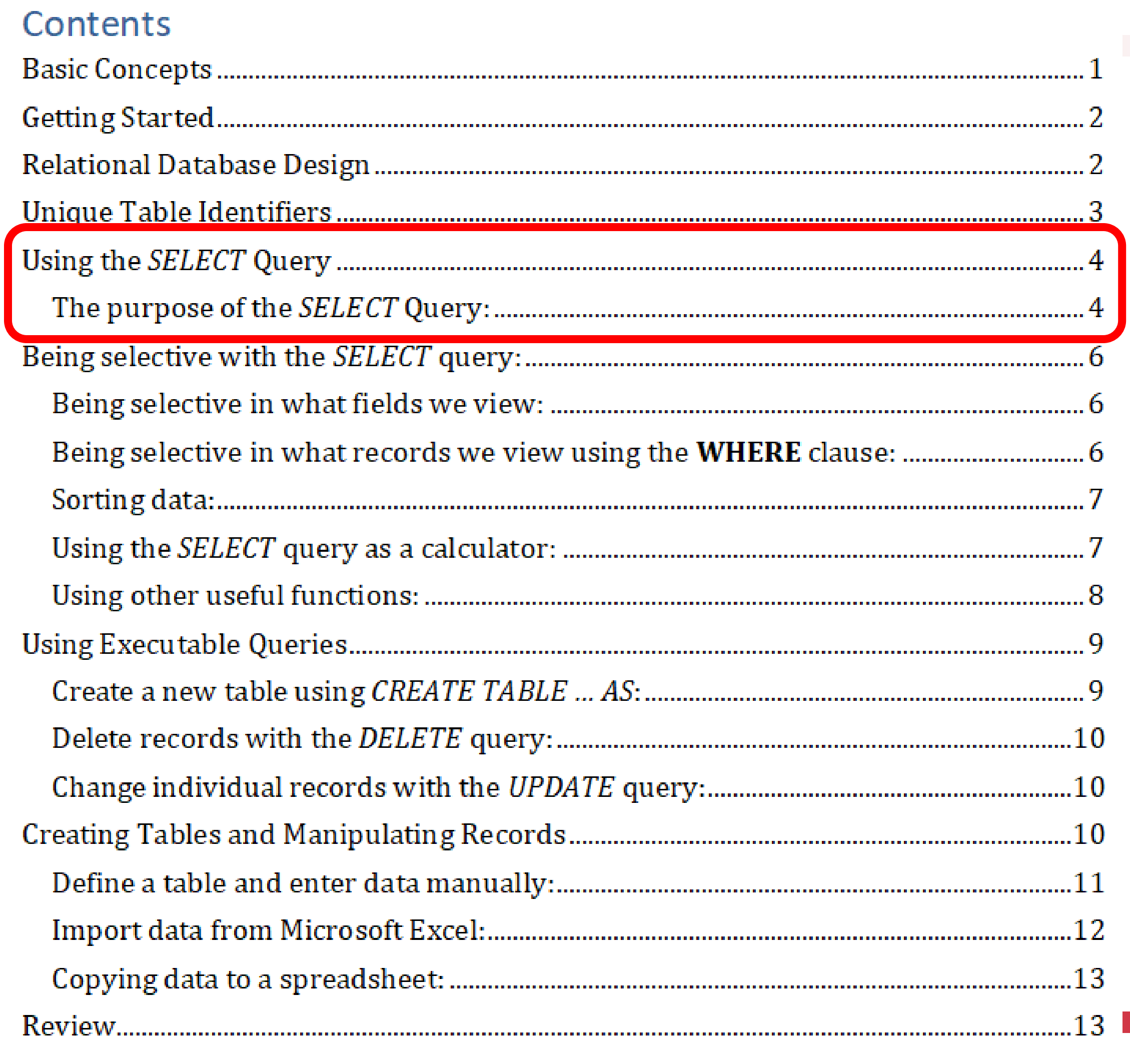
	InvoiceId	CustomerI(	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostal(	Total
1	1	2	2009-01-01 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	NULL	Germany	70174	1.98
2	2	4	2009-01-02 00:00:00	Ullevålsveien 14	Oslo	NULL	Norway	0171	3.96
3	3	8	2009-01-03 00:00:00	Grétrystraat 63	Brussels	NULL	Belgium	1000	5.94
4	4	14	2009-01-06 00:00:00	8210 111 ST NW	Edmonton	AB	Canada	T6G 2C7	8.91
5	5	23	2009-01-11 00:00:00	69 Salem Street	Boston	MA	USA	2113	13.86
6	6	37	2009-01-19 00:00:00	Berger Straße 10	Frankfurt	NULL	Germany	60316	0.99
7	7	38	2009-02-01 00:00:00	Barbarossastraße 19	Berlin	NULL	Germany	10779	1.98
8	8	40	2009-02-01 00:00:00	8, Rue Hanovre	Paris	NULL	France	75002	1.98
9	9	42	2009-02-02 00:00:00	9, Place Louis Barthou	Bordeaux	NULL	France	33000	3.96
10	10	46	2009-02-03 00:00:00	3 Chatham Street	Dublin	Dublin	Ireland	NULL	5.94
11	11	52	2009-02-06 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	8.91
12	12	2	2009-02-11 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	NULL	Germany	70174	13.86
13	13	16	2009-02-19 00:00:00	1600 Amphitheatre Parkway	Mountain View	CA	USA	94043-1351	0.99
14	14	17	2009-03-04 00:00:00	1 Microsoft Way	Redmond	WA	USA	98052-8300	1.98
15	15	19	2009-03-04 00:00:00	1 Infinite Loop	Cupertino	CA	USA	95014	1.98

Looks like a table, can be treated like a table (sometimes), but exists only in temporary memory

You could save the query as a "*view*"

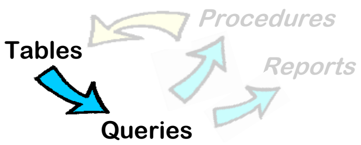
OR you could create a view first and then input the query

SQLite Activity



Contents

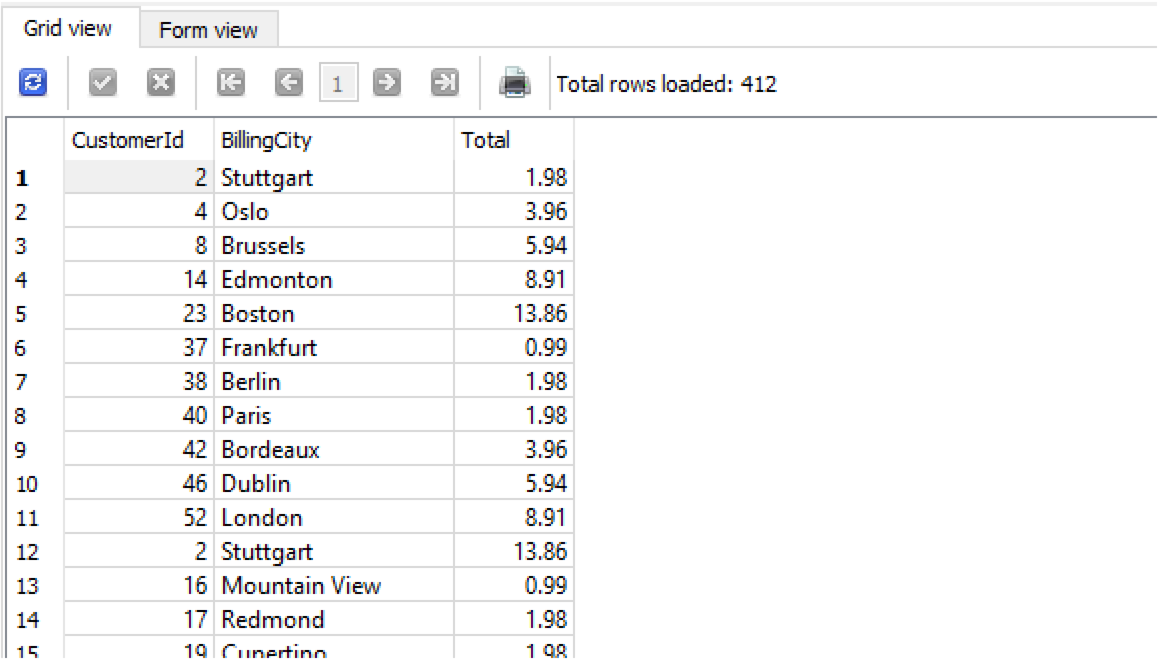
Select Only the Fields You are Interested In



E.g., you are interested in customers, their billing cities and their bills

```
SELECT CustomerId, BillingCity, Total FROM Invoice
```

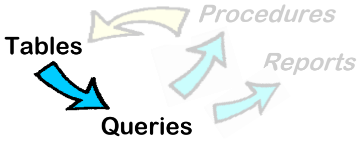
Query results



Total rows loaded: 412

	CustomerId	BillingCity	Total
1	2	Stuttgart	1.98
2	4	Oslo	3.96
3	8	Brussels	5.94
4	14	Edmonton	8.91
5	23	Boston	13.86
6	37	Frankfurt	0.99
7	38	Berlin	1.98
8	40	Paris	1.98
9	42	Bordeaux	3.96
10	46	Dublin	5.94
11	52	London	8.91
12	2	Stuttgart	13.86
13	16	Mountain View	0.99
14	17	Redmond	1.98
15	19	Cupertino	1.98

Select Only the Records You are Interested In

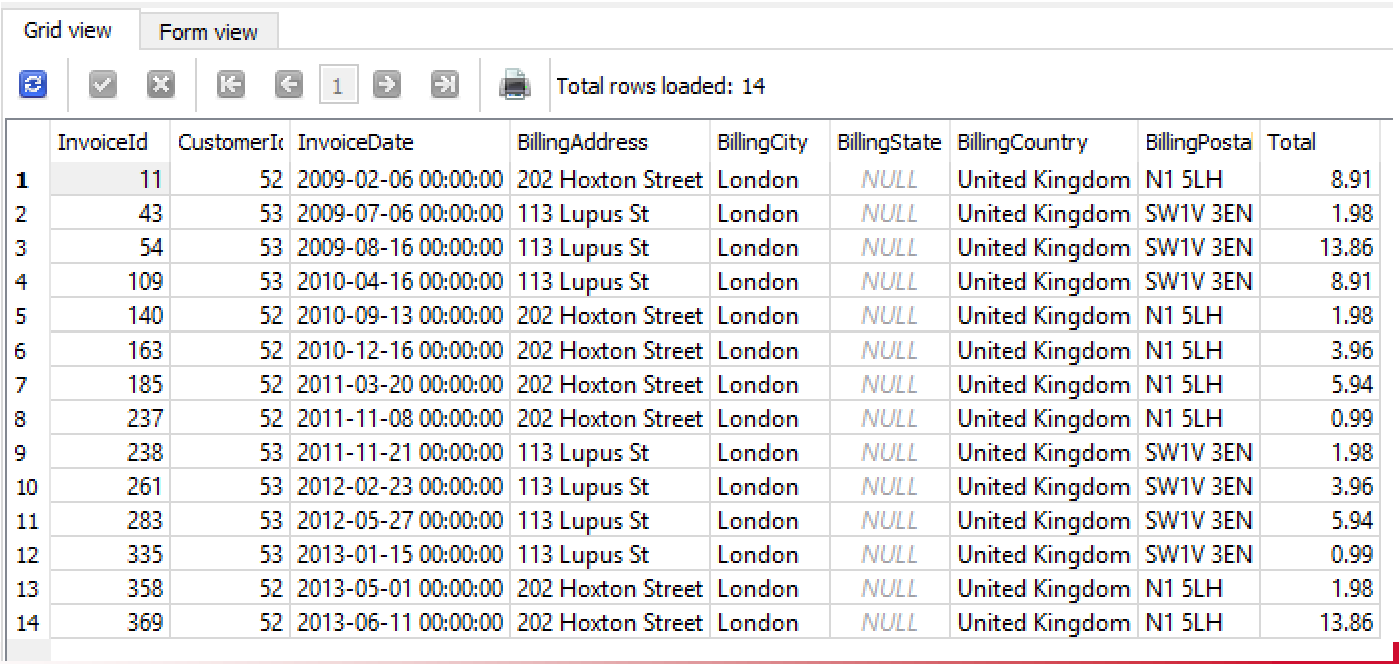


E.g., you are interested in orders from one city, say “London”

```
SELECT * FROM Invoice WHERE BillingCity = 'London'
```

WHERE clause describes selection criteria

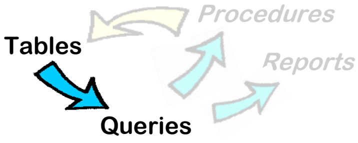
Query results



Total rows loaded: 14

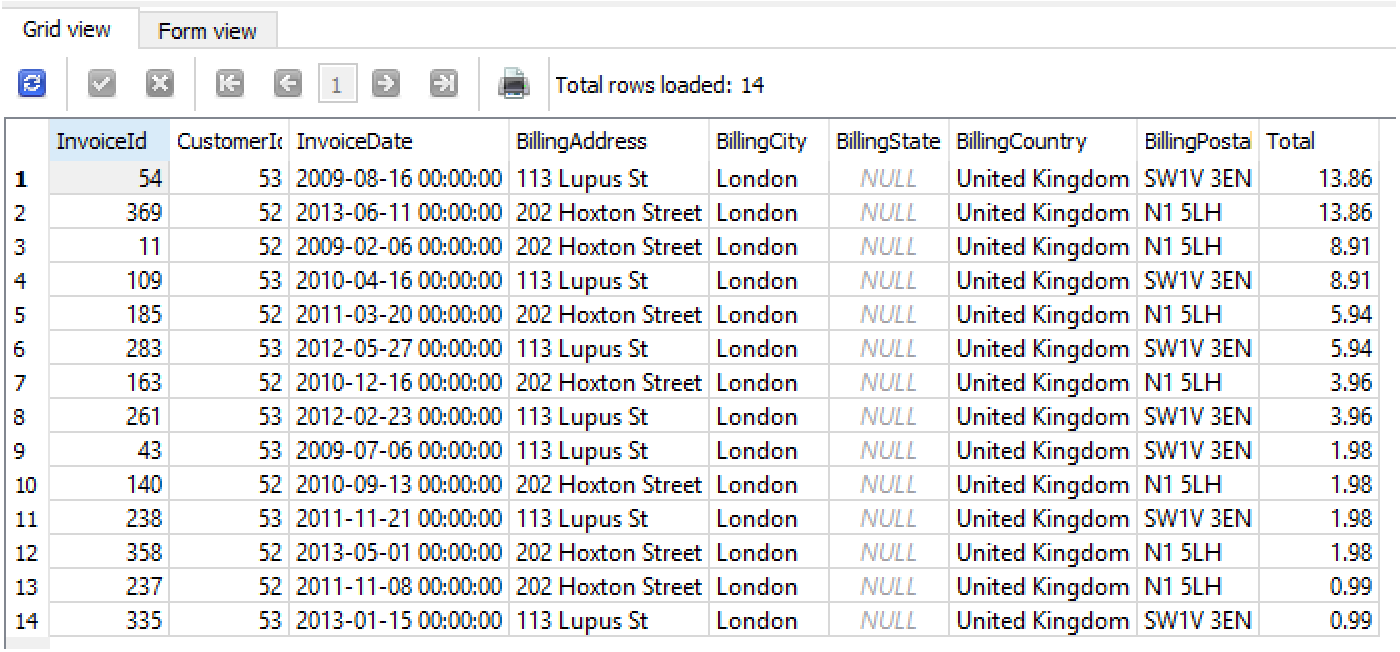
	InvoiceId	CustomerI	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPosta	Total
1	11	52	2009-02-06 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	8.91
2	43	53	2009-07-06 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	1.98
3	54	53	2009-08-16 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	13.86
4	109	53	2010-04-16 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	8.91
5	140	52	2010-09-13 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	1.98
6	163	52	2010-12-16 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	3.96
7	185	52	2011-03-20 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	5.94
8	237	52	2011-11-08 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	0.99
9	238	53	2011-11-21 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	1.98
10	261	53	2012-02-23 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	3.96
11	283	53	2012-05-27 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	5.94
12	335	53	2013-01-15 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	0.99
13	358	52	2013-05-01 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	1.98
14	369	52	2013-06-11 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	13.86

Select Only the Records You are Interested In



E.g., you are interested in orders from London and Need to sort them by the total amount of bill from high to low

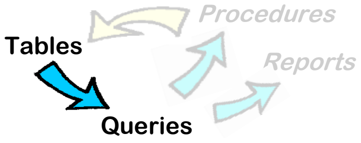
```
SELECT * FROM Invoice WHERE BillingCity = ‘London’ ORDER BY Total DESC
```



Total rows loaded: 14

	InvoiceId	CustomerI	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPosta	Total
1	54	53	2009-08-16 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	13.86
2	369	52	2013-06-11 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	13.86
3	11	52	2009-02-06 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	8.91
4	109	53	2010-04-16 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	8.91
5	185	52	2011-03-20 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	5.94
6	283	53	2012-05-27 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	5.94
7	163	52	2010-12-16 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	3.96
8	261	53	2012-02-23 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	3.96
9	43	53	2009-07-06 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	1.98
10	140	52	2010-09-13 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	1.98
11	238	53	2011-11-21 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	1.98
12	358	52	2013-05-01 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	1.98
13	237	52	2011-11-08 00:00:00	202 Hoxton Street	London	NULL	United Kingdom	N1 5LH	0.99
14	335	53	2013-01-15 00:00:00	113 Lupus St	London	NULL	United Kingdom	SW1V 3EN	0.99

Use Select as a Calculator

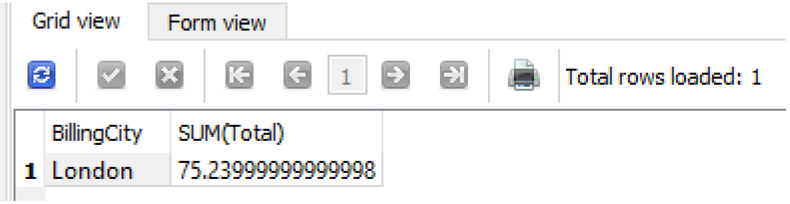


E.g., you are interested in total amount of bills paid by all the customer from London

```
SELECT BillingCity, SUM(Total)  FROM Invoice WHERE BillingCity = 'London'
```

SUM clause returns the total sum of a numeric column

Query results



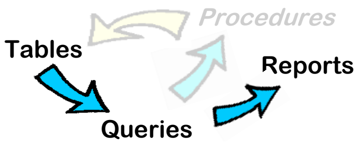
Total rows loaded: 1

	BillingCity	SUM(Total)
1	London	75.23999999999998

Note: you could use AS clause to specify the name of the new field calculated

```
SELECT BillingCity, SUM(Total) AS CustomerTotal
FROM Invoice WHERE BillingCity = 'London'
```

Aggregate Data Query



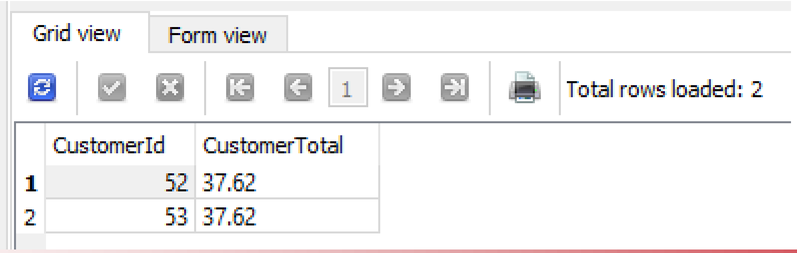
Aggregate functions include: SUM(), COUNT(), MAX(), MIN(), AVG(), STDEV(), FIRST(), LAST()

Aggregate Query with Sub-Totals: Use GROUP BY statement with aggregate functions to group the result-set by one or more columns.

E.g., you are interested in total amount of bills of each customer from London

```
SELECT CustomerId, SUM(Total) AS CustomerTotal
FROM Invoice WHERE BillingCity = 'London‘ GROUP BY CustomerId;
```

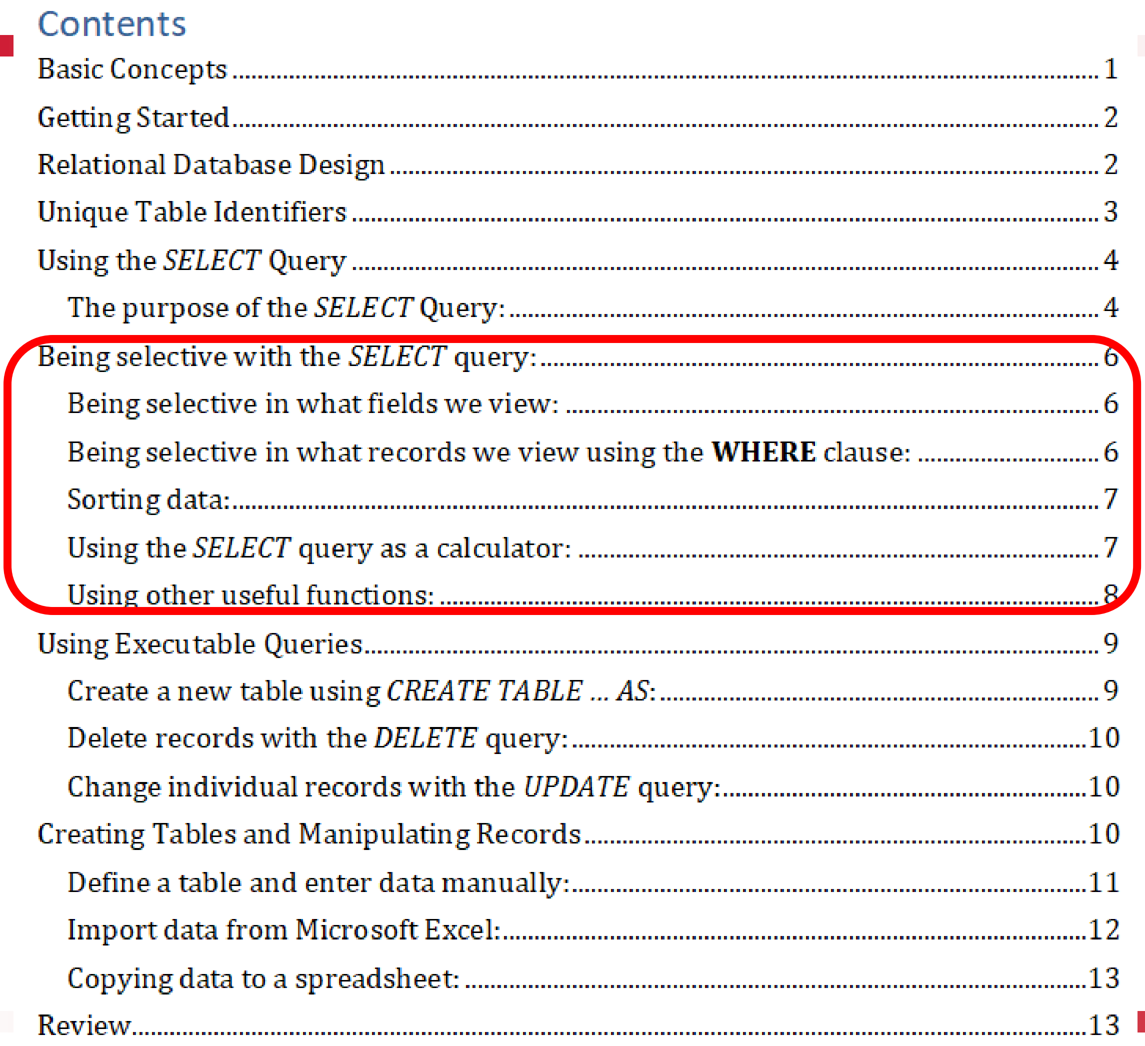
Query results



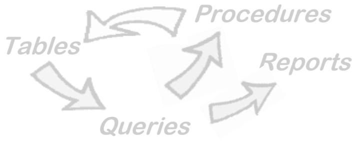
Total rows loaded: 2

	CustomerId	CustomerTotal
1	52	37.62
2	53	37.62

SQLite Activity



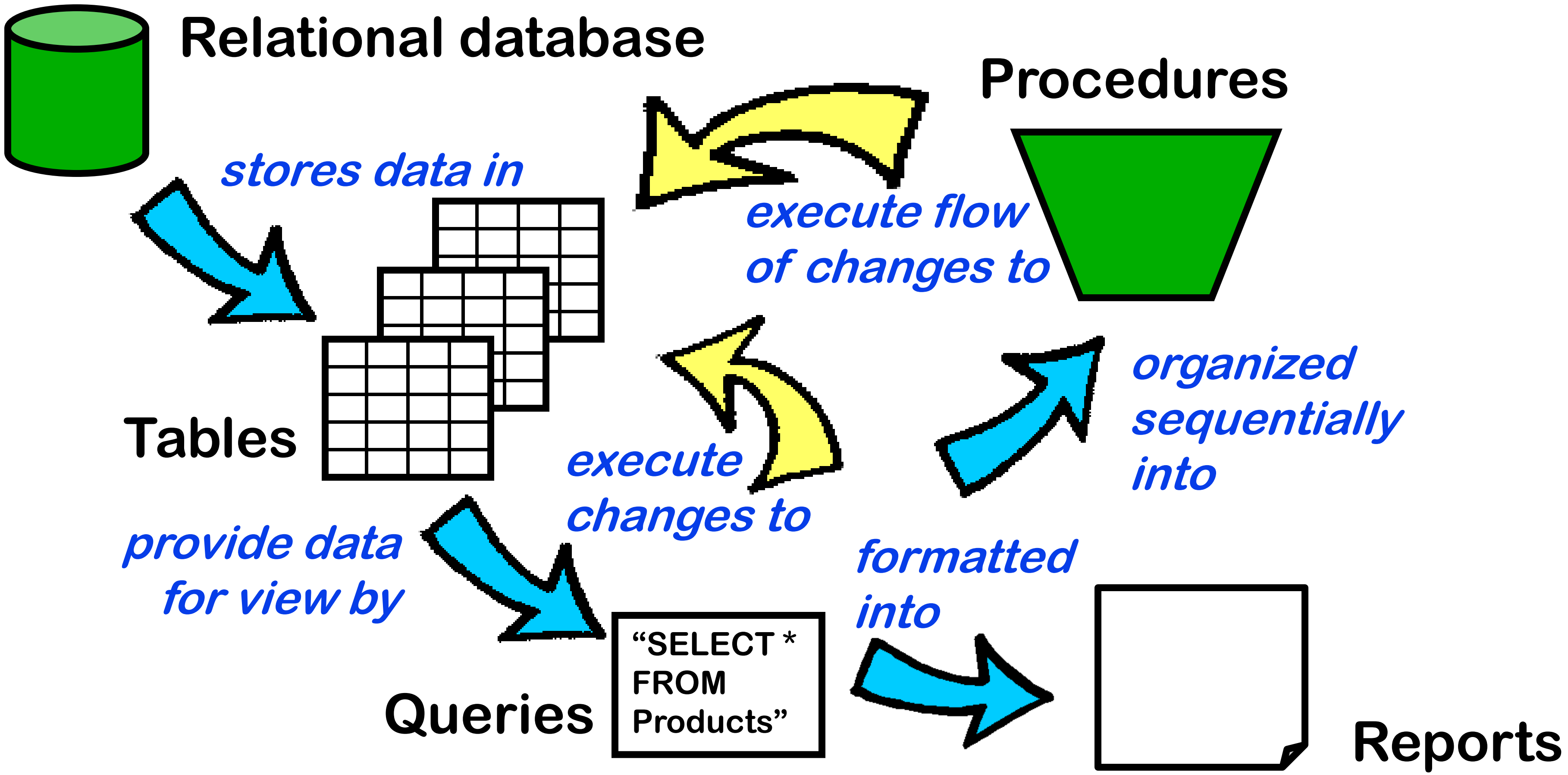
Contents



Overview

- ✓ Tables
  - Sample Table with Key
  - Use Keys to Make Relationships
  - Tables are Related by Keys
- ✓ Queries
  - Query Result
- ✓ Save the Query
- ✓ Selective Query Result
  - Select Only the Fields You are Interested In
  - Select Which Records You are interested in
- ✓ Use Select as a Calculator
- ✓ Aggregate Data Query
  - Aggregate Query with Sub-Totals
- Make a New Table
- Modify Data in the Database

Relational Database Management System (RDBMS)



Relational database

stores data in

Tables

provide data for view by

Queries

“SELECT * FROM Products”

execute changes to

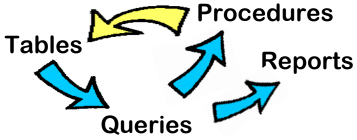
execute flow of changes to

Procedures

organized sequentially into

formatted into

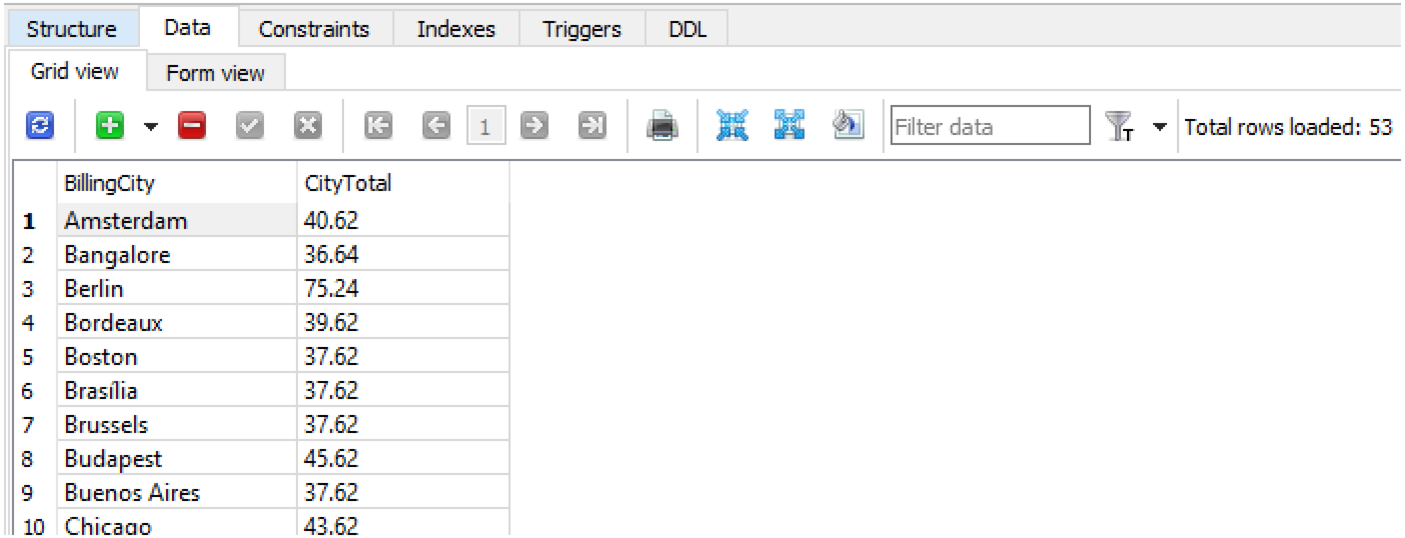
Reports



Make a New Table

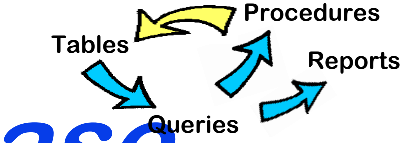
You are interested in creating a new table recording the total amount of bills paid by customers in each city

CREATE TABLE RevenueCity AS SELECT BillingCity, SUM(Total) AS CityTotal FROM Invoice GROUP BY BillingCity



Total rows loaded: 53

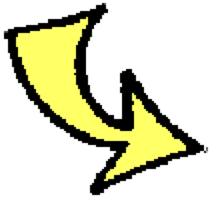
	BillingCity	CityTotal
1	Amsterdam	40.62
2	Bangalore	36.64
3	Berlin	75.24
4	Bordeaux	39.62
5	Boston	37.62
6	Brasília	37.62
7	Brussels	37.62
8	Budapest	45.62
9	Buenos Aires	37.62
10	Chicago	43.62

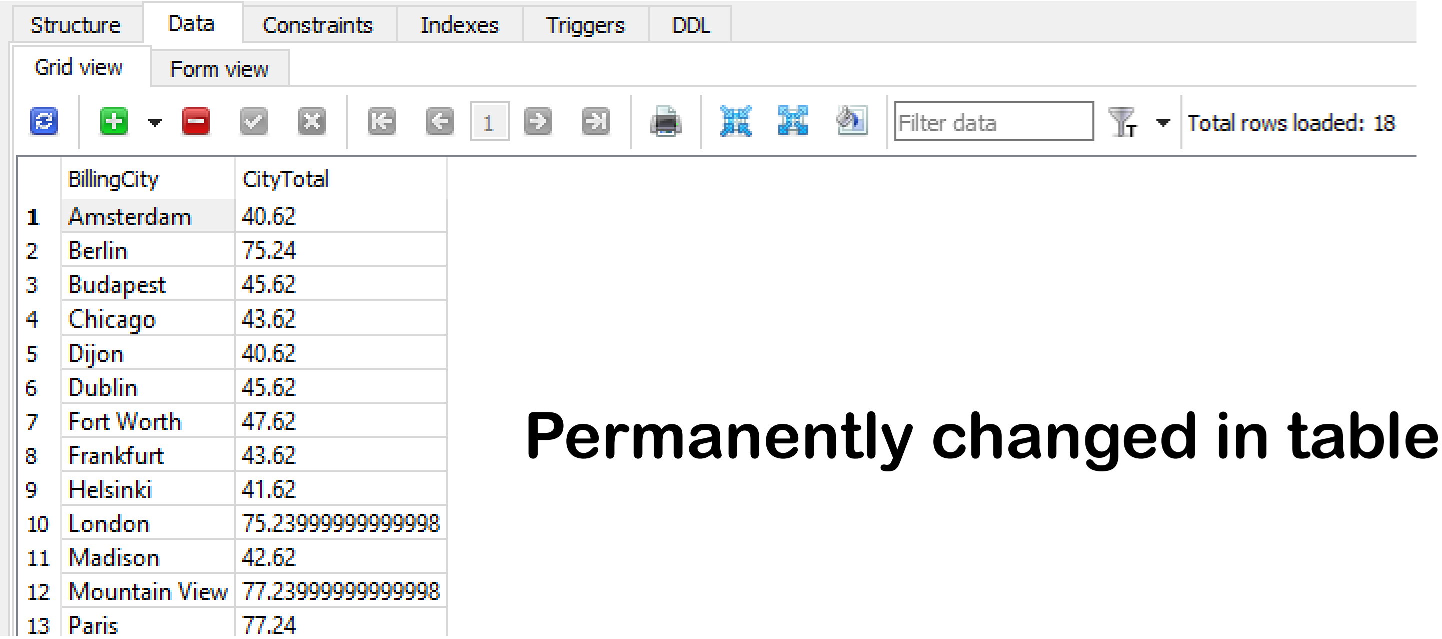


Modify Data in the Database

You are only interested in those cities who have a total revenues more than 40

```
DELETE FROM RevenueCity WHERE CityTotal <= 40
```



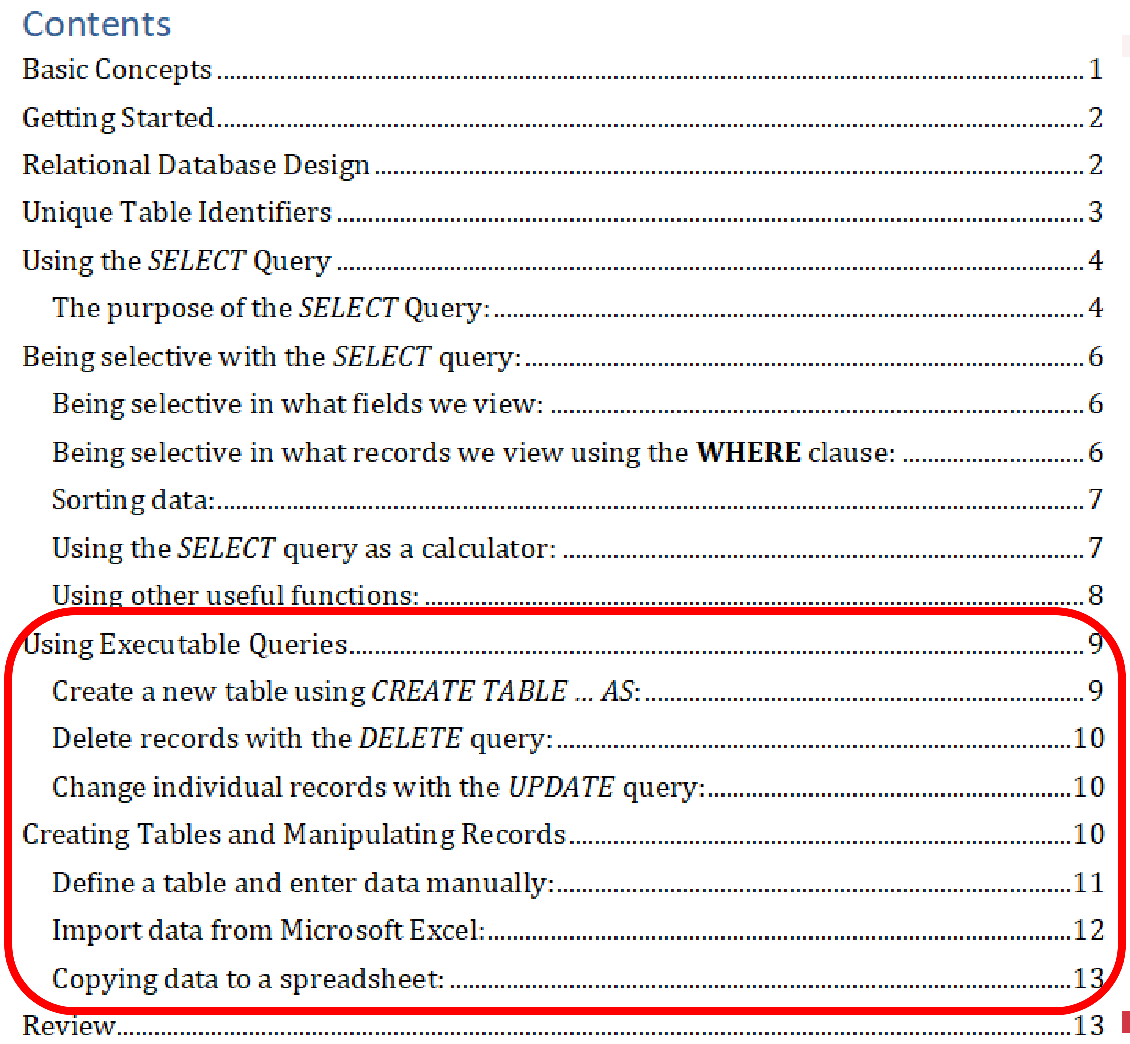


Total rows loaded: 18

	BillingCity	CityTotal
1	Amsterdam	40.62
2	Berlin	75.24
3	Budapest	45.62
4	Chicago	43.62
5	Dijon	40.62
6	Dublin	45.62
7	Fort Worth	47.62
8	Frankfurt	43.62
9	Helsinki	41.62
10	London	75.2399999999998
11	Madison	42.62
12	Mountain View	77.2399999999998
13	Paris	77.24

Permanently changed in table

SQLite Activity



Contents