What is a view?

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, ...	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry Ln,	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs,	300	50000	30002	5
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora,	400	60000	30004	5

views.

What is a view?

EMP_ID	F_NAME	L_NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1001	John	Thomas	5631 Rice, ...	100	30001	2
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5
E1004	Santosh	Kumar	511 Aurora,	400	30004	5

- Can include specified columns from multiple base tables and existing views
- Once created, can be queried like a table
- Only the definition of the view is stored, not the data

The data that the view represents is stored in the base tables, not by the view itself.

When should you use a view?

Views can:

- Show a selection of data for a given table
- Combine two or more tables in meaningful ways
- Simplify access to data
- Show only portions of data in the table

Example:

- Create a view to display non-sensitive data from the Employees table

CREATE VIEW statement

```
CREATE VIEW <view name> (<column_alias_1>,
<column_alias_2>, … <column_alias_n>)
  AS SELECT <column_1> , <column_2>, … <column_n>
  FROM <table name>
  WHERE <predicate>;
```

You can also add an optional WHERE clause to refine the rows in the view.

CREATE VIEW statement

```
CREATE VIEW EMPINFO (EMP_ID, FIRSTNAME, LASTNAME, ADDRESS,
JOB_ID, MANAGER_ID, DEP_ID)
  AS SELECT EMP_ID, F_NAME, L_NAME, ADDRESS, JOB_ID,
MANAGER_ID, DEP_ID
  FROM EMPLOYEES;
```

```
SELECT * FROM EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1001	John	Thomas	5631 Rice, ...	100	30001	2
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5
E1004	Santosh	Kumar	511 Aurora,	400	30004	5

Working with views

```
CREATE VIEW EMPINFO (EMP_ID, FIRSTNAME, LASTNAME, ADDRESS,
JOB_ID, MANAGER_ID, DEP_ID)
  AS SELECT EMP_ID, F_NAME, L_NAME, ADDRESS, JOB_ID,
MANAGER_ID, DEP_ID
FROM EMPLOYEES
WHERE MANAGER_ID = '30002'
```

```
SELECT * FROM EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5

Working with views

```
DROP VIEW EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5

To exit full screen, press Esc

What is a stored procedure?

- A set of SQL statements stored and executed on the database server
  - Write in many different languages
  - Accept information in the form of parameters
  - Return results to the client

and return results to
the client application.

Benefits of stored procedures

- Reduction in network traffic
- Improvement in performance
- Reuse of code
- Increase in security

You should not try
to write all of

CREATE PROCEDURE statement

```
DELIMITER $$
CREATE PROCEDURE UPDATE_SAL (IN empNum CHAR(6), IN rating
SMALLINT)
BEGIN
UPDATE employees SET salary = salary * 1.10 WHERE emp_id
= empNum AND rating = 1;
UPDATE employees SET salary = salary * 1.05 WHERE emp_id
= empNum AND rating <> 1;
END
$$
DELIMITER ;
```

back to semicolon
when we are done.

Working with stored procedures

Call from:

- External applications
- Dynamic SQL statements

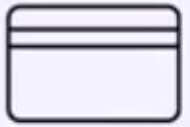
```
CALL UPDATE_SAL ('E1001', 1)
```

the stored procedure and pass
the required parameters.

What is a transaction?

- Indivisible unit of work
- Consists of one or more SQL statements
- Either all happens or none





The inventory for that product must be reduced by the number purchased.

Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE BankAccounts
SET Balance = Balance-200
WHERE AccountName = 'Rose';
```

Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE BankAccounts
SET Balance = Balance+200
WHERE AccountName = 'Shoe Shop';
```

Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE ShoeShop
SET Stock = Stock-1
WHERE Product = 'Boots';
```

What is an ACID transaction?

Atomic

All changes must be performed successfully or not at all.

Consistent

Data must be in a consistent state before and after the transaction.

Isolated

No other process can change the data while the transaction is running.

Durable

The changes made by the transaction must persist.

ACID commands

- BEGIN
  - Start the ACID transaction
- COMMIT
  - All statements complete successfully
  - Save the new database state
- ROLLBACK
  - One or more statements fail
  - Undo changes

```
BEGIN
        UPDATE BankAccounts
        SET Balance = Balance – 200
        WHERE AccountName = 'Rose';

        UPDATE BankAccounts
        SET Balance = Balance + 200
        WHERE AccountName = 'Shoe Shop';

        UPDATE ShoeShop
        SET Stock = Stock - 1
        WHERE Product = 'Boots';
ROLLBACK
```

Calling ACID commands

- Can also be issued by some languages
  - Java, C, R, and Python
  - Requires the use of database specific APIs or connectors
- Use the EXEC SQL command to execute SQL statements from code

```
void main ()
  {
      EXEC SQL  UPDATE BankAccounts
        SET Balance = Balance – 200
        WHERE AccountName = 'Rose';
      EXEC SQL  UPDATE BankAccounts
        SET Balance = Balance + 200
        WHERE AccountName = 'Shoe Shop';
  EXEC SQL UPDATE ShoeShop
        SET Stock = Stock - 1
        WHERE Product = 'Boots';
      FINISHED:
      EXEC SQL  COMMIT WORK;
      return;

    SQLERR:
      EXEC SQL  WHENEVER SQLERROR CONTINUE;
      EXEC SQL  ROLLBACK WORK;
      return;
  }
```

Summary

In this video, you learned that:

- A transaction is a unit of work which usually consists of multiple SQL statements
- In an ACID transaction all the SQL statements must complete successfully, or none at all
- ACID stands for Atomic, Consistent, Isolated, Durable
- BEGIN, COMMIT, ROLLBACK
- Can be called from languages like C, R and Python

Why Data Science and SQL

- A data-centric world requires data science
- Vast data repositories

Why Data Science and SQL

SQL helps

Query data

Update data

Manipulate data

SQL and Python

- Programming language
- Connects databases together
- Execute SQL queries
- Wide array of libraries

What is SQL?

A language used for relational databases

Query data

What is data?

- Facts (words, numbers)
- Pictures
- One of the most critical assets of any business
- Needs to be secure

What is a database?

A repository of data

Provides the functionality for adding, modifying and querying that data

Different kinds of databases store data in different forms

Relational Database

Student ID	First Name	Last Name
34933	Victoria	Slater
93759	Justin	McNeil
20847	Jessica	Bennett
65947	Michelle	Dolin
24956	David	Price
65692	Franklin	Mullins
24271	Alissa	Lee

- Data stored in tabular form - columns and rows
- Columns contain item properties e.g. Last Name, First Name, etc.
- Table is collection of related things e.g. Employees, Authors, etc.
- Relationships can exist between tables (hence: "relational")

DBMS

Database: repository of data

DBMS: Database Management System - software to manage databases

Database, Database Server, Database System, Data Server, DBMS - often used interchangeably

What is RDBMS?

- RDBMS = Relational database management system
- A set of software tools that controls the data
  - Access, organization, and storage
- Examples are: MySQL, Oracle Database, IBM Db2, etc.

Basic SQL Commands

Create a table

Insert

Select

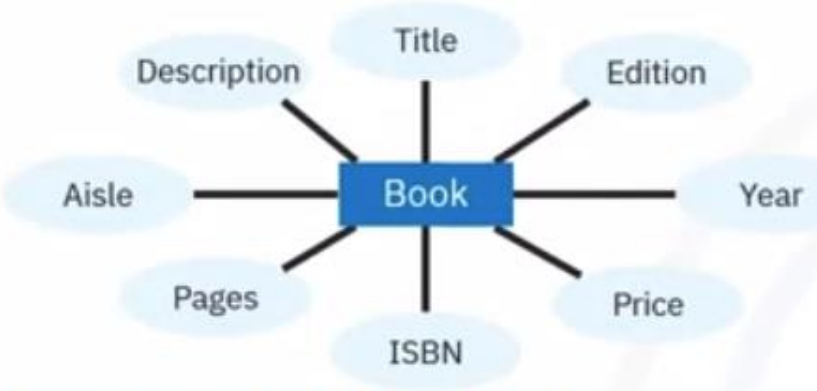
Update

Retrieving rows from a table

- After creating a table and inserting data into the table, we want to see the data
- SELECT statement
  - A Data Manipulation Language (DML) statement used to read and modify data

```
Select statement: Query
Result from the query:  Result set/table

Select * from <tablename>
```

Using the SELECT Statement



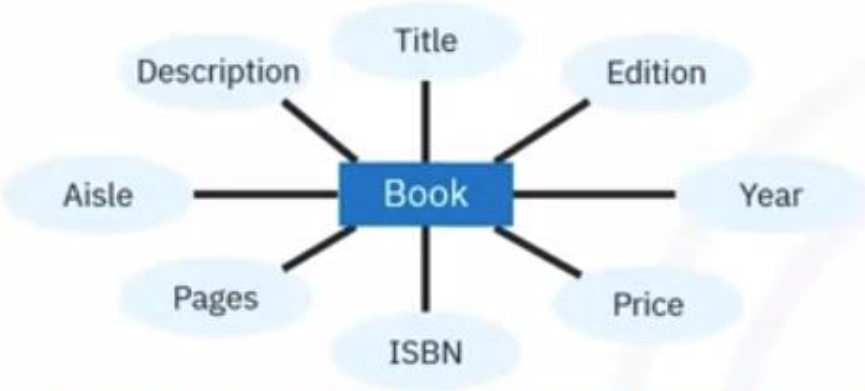
Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-9800628 3-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866628 3-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C

Example: select * from Book

```
db2 => select * from Book
```

Book_ID	Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
B1	Getting started with DB2 Express-C	1	2010	24.99	978-0-98666283-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C
B2	Database Fundamentals	1	2010	24.99	978-0-98006283-1-1	300	DB-A02	Teaches you the fundamentals of databases
B3	Getting started with DB2 App Dev	1	2011	35.99	978-0-98086283-4-1	345	DB-A03	Teaches you the essentials of developing applications for DB2.
B4	Getting started with WAS CE	1	2010	49.99	978-0-98946283-3-1	458	DB-A04	Teaches you the essentials of WebSphere Application Server
4 record(s) selected.								

Using the SELECT Statement



Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-9800628 3-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866628 3-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C

Example: select <column 1, column 2, ..., column n from Book

```
db2 => select book_id, title, edition, year, price, ISBN, pages, aisle, description from
```

Book_ID	Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
B1	Getting started with DB2 Express-C	1	2010	24.99	978-0-98666283-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C
B2	Database Fundamentals	1	2010	24.99	978-0-98006283-1-1	300	DB-A02	Teaches you the fundamentals of databases
B3	Getting started with DB2 App Dev	1	2011	35.99	978-0-98086283-4-1	345	DB-A03	Teaches you the essentials of developing applications for DB2.
B4	Getting started with WAS CE	1	2010	49.99	978-0-98946283-3-1	458	DB-A04	Teaches you the essentials of WebSphere Application Server
4 record(s) selected.								

Retrieving a subset of the columns

- You can retrieve just the columns you want
- The order of the columns displayed always matches the order in the SELECT statement
  - SELECT <column 1>, <column 2> from Book

```
db2 => select book_id, title from Book
```

Book_ID	Title
B1	Getting started with DB2 Express-C
B2	Database Fundamentals
B3	Getting started with DB2 App Dev
B4	Getting started with WAS CE
4 record(s) selected.	

Restricting the Result Set: WHERE Clause

- Restricts the result set
- Always requires a Predicate:
  - Evaluates to: True, False, or Unknown
  - Used in the search condition of the Where clause

```
select book_id, title from Book
       WHERE predicate
```

```
db2 => select book_id, title from Book
       WHERE book_id='B1'
```

Book_ID	Title
B1	Getting started with DB2 Express-C
1 record(s) selected	

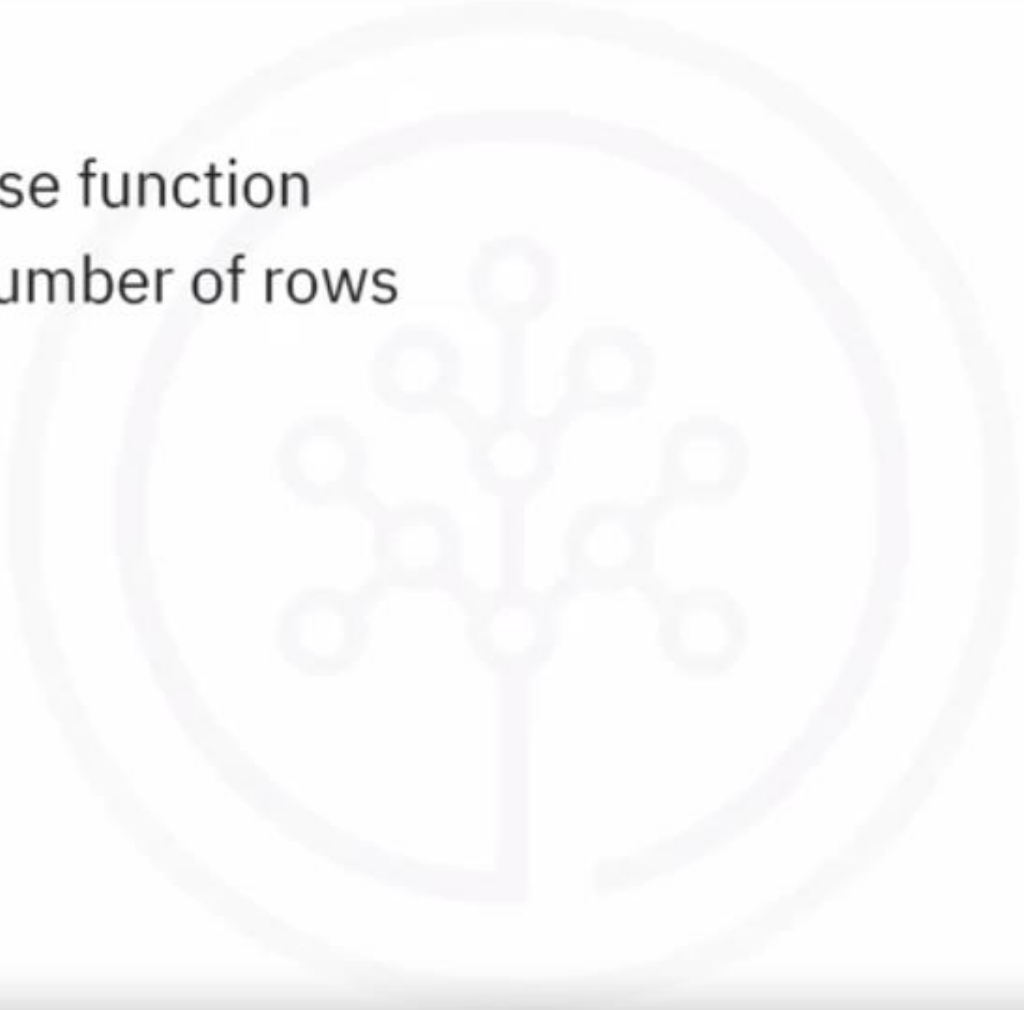
WHERE Clause Comparison Operators

```
select book_id, title from Book
    WHERE book_id = 'B1'
```

Equal to	=
Greater than	>
Lesser than	<
Greater than or equal to	>=
Less than or equal to	<=
Not equal to	<>

COUNT

1. **Count**
    - built-in database function
    - retrieves the number of rows



COUNT

Example:

MEDALS

Awardees	Country	Awards	Year
John	Canada	Gold	2023
Sam	India	Gold	2023
Peter	Canada	Silver	2022
Aby	Australia	Gold	2023
Richard	Canada	Bronze	2023
David	Canada	Gold	2023

COUNT

Example:

```
select COUNT(COUNTRY) from MEDALS
       where COUNTRY='CANADA'
```

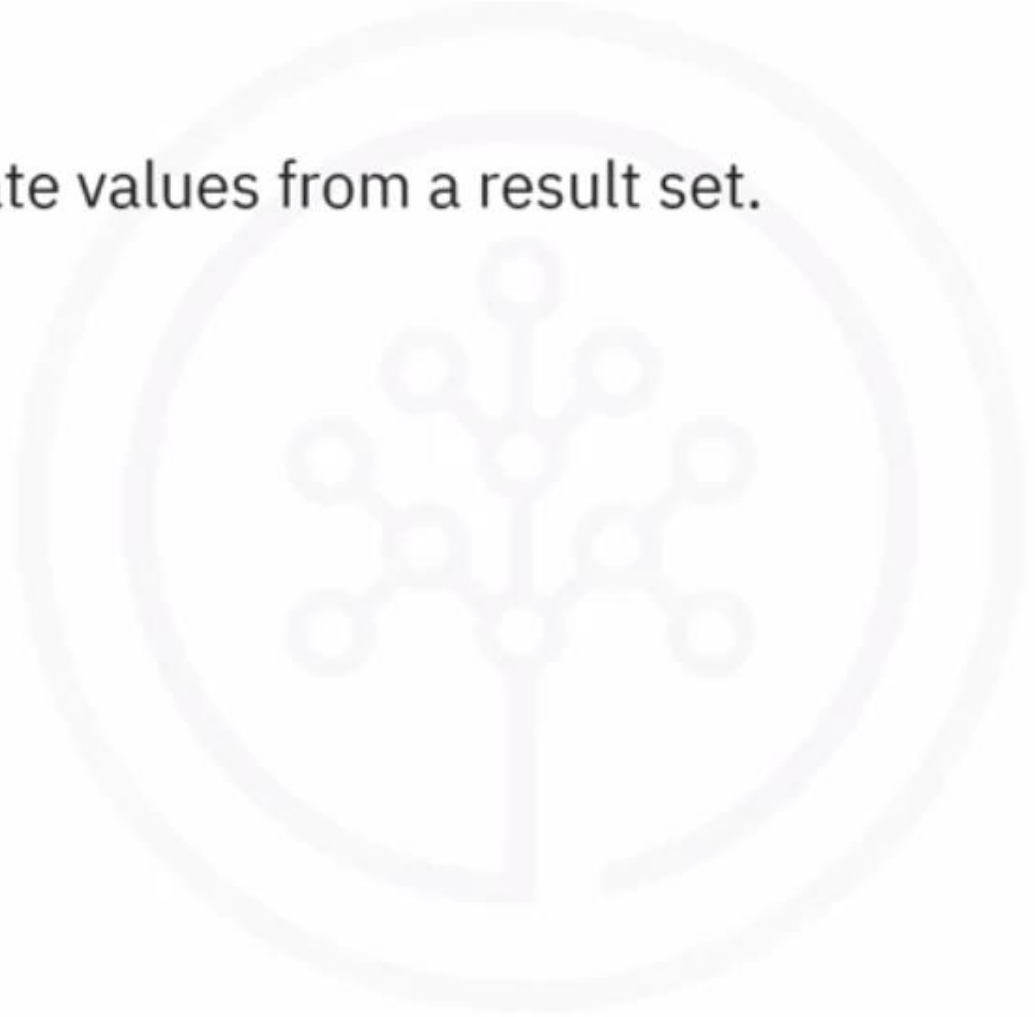
Result:

1
29

DISTINCT

2. DISTINCT

Removes duplicate values from a result set.



DISTINCT

Retrieves unique values in a column:

```
select DISTINCT columnname from tablename
```

DISTINCT

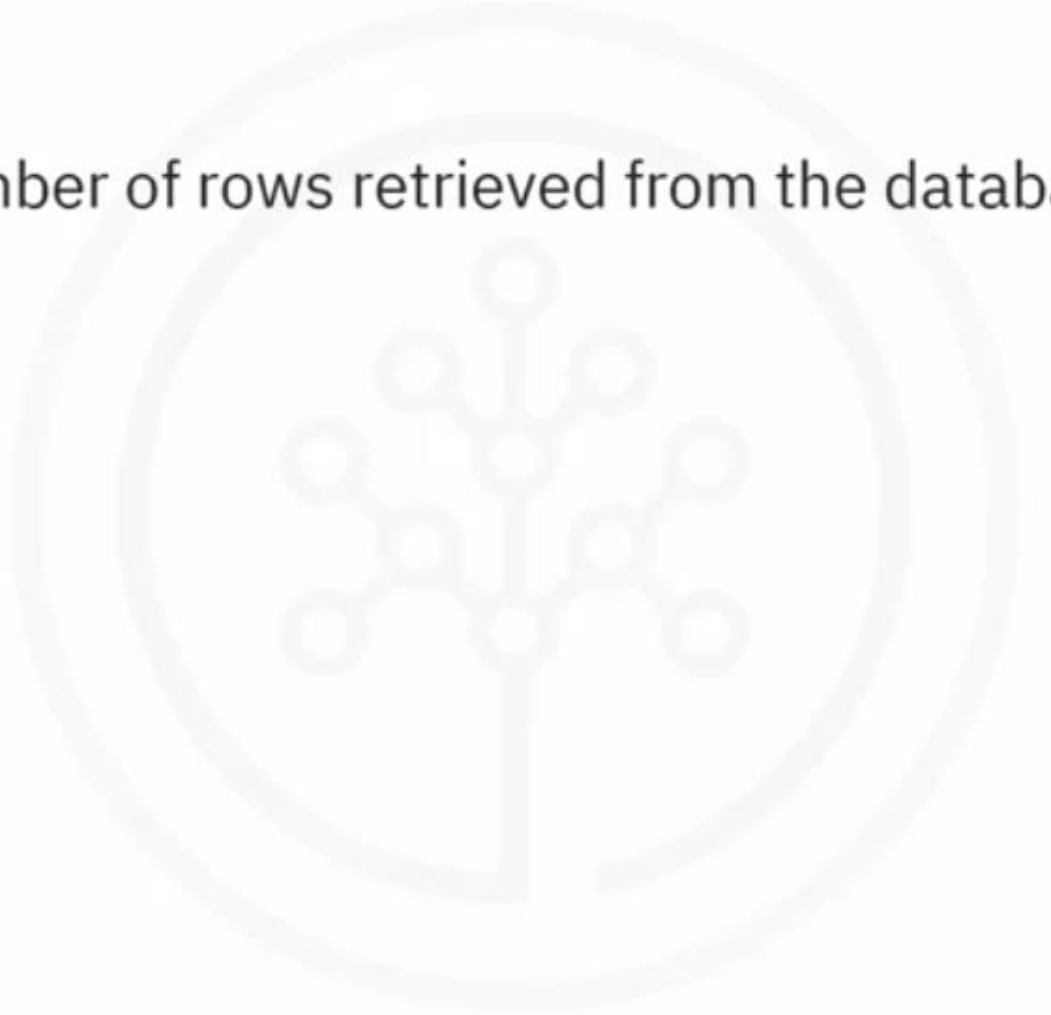
List of unique countries that received GOLD medals:

```
select DISTINCT COUNTRY from MEDALS
        where MEDALTYPE = 'GOLD'
```

LIMIT

3. LIMIT

Restricts the number of rows retrieved from the database.



LIMIT

Retrieve just the first 10 rows in a table:

```
select * from tablename LIMIT 10
```

LIMIT

Retrieve 5 rows in the MEDALS table for a particular year:

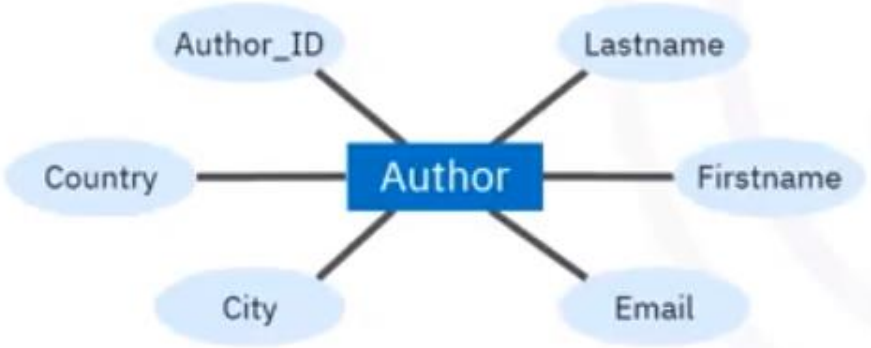
```
select * from MEDALS
        where YEAR = 2018 LIMIT 5
```

Result:

COUNTRY	GOLD	SILVER	BRONZE	TOTAL	YEAR
Norway	14	14	11	39	2018
Germany	14	10	7	31	2018
Canada	11	8	10	29	2018
United States	9	8	6	23	2018
Netherlands	8	6	6	20	2018

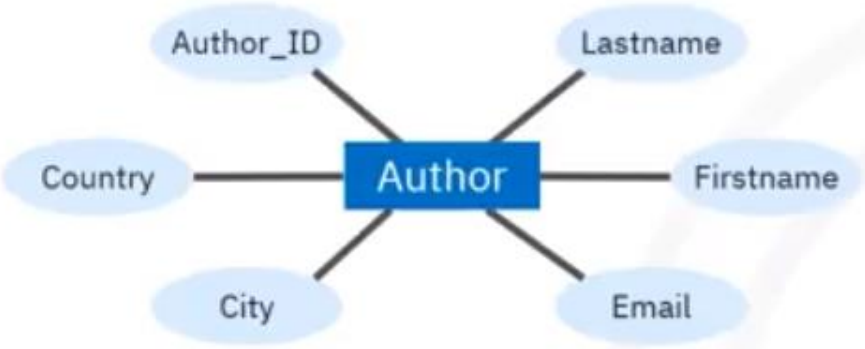
Adding rows to a table

- Create the table (CREATE TABLE statement)
- Populate table with data:
  - INSERT statement
    - a Data Manipulation Language (DML) statement used to read and modify data



Author_ID	Lastname	Firstname	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	Ca
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

Using the INSERT Statement

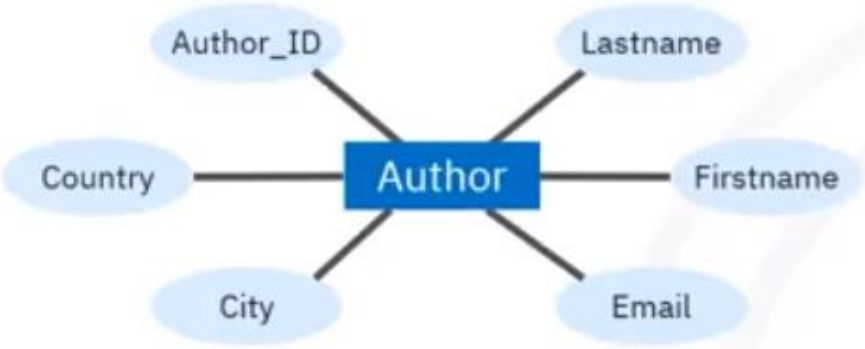


Author_ID	Lastname	Firstname	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	Ca
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

```
INSERT INTO [TableName]

<([ColumnName],…)>
VALUES ([Value],…)
```

```
INSERT INTO AUTHOR
      (AUTHOR_ID, LASTNAME, FIRSTNAME, EMAIL,
CITY, COUNTRY)
VALUES ('A1', 'Chong', 'Raul', 'rfc@ibm.com',
'Toronto', 'CA')
```

Inserting multiple rows



Author_ID	Lastname	Firstname	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	Ca
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

```
INSERT INTO AUTHOR
      (AUTHOR_ID, LASTNAME, FIRSTNAME, EMAIL, CITY, COUNTRY)
VALUES
      ('A1', 'Chong', 'Raul', 'rfc@ibm.com', 'Toronto', 'CA'),
      ('A2', 'Ahuja', 'Rav', 'ra@ibm.com', 'Toronto', 'CA')
```

Altering rows of a table – UPDATE statement

Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

Altering rows of a table – UPDATE statement

UPDATE statement

1. DML statements
2. Read and modify data

Using the UPDATE statement

UPDATE [TableName] SET [[ColumnName]=[Value]] <WHERE [Condition]>

Using the UPDATE statement

Author_Id	LastName	FirstName	Email	City	Country
A1	CHONG	RAUL	rfc@ibm.com	Toronto	CA
A2	AHUJA	RAV	ra@ibm.com	Toronto	CA
A3	HAKES	IAN	ih@ibm.com	Toronto	CA

Using the UPDATE statement

```
UPDATE AUTHOR SET LASTNAME='KATTA’ FIRSTNAME='LAKSHMI’
WHERE AUTHOR_ID='A2'
```

Using the UPDATE statement

Author_Id	LastName	FirstName	Email	City	Country
A1	CHONG	RAUL	rfc@ibm.com	Toronto	CA
A2	KATTA	LAKSHMI	ra@ibm.com	Toronto	CA
A3	HAKES	IAN	ih@ibm.com	Toronto	CA

Using the DELETE statement

Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

```
DELETE FROM AUTHOR
    WHERE AUTHOR_ID IN ('A2', 'A3')
```

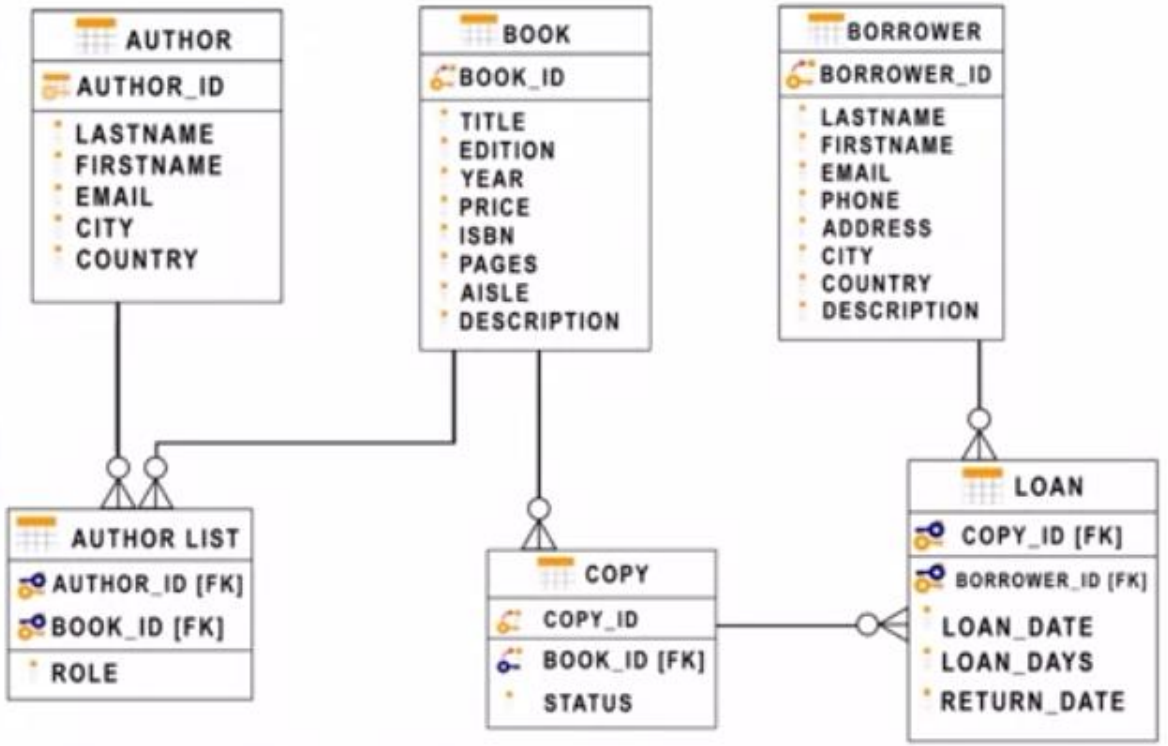
Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

If no WHERE clause is used, all rows will be removed

Relational Database Concepts

Relational Model

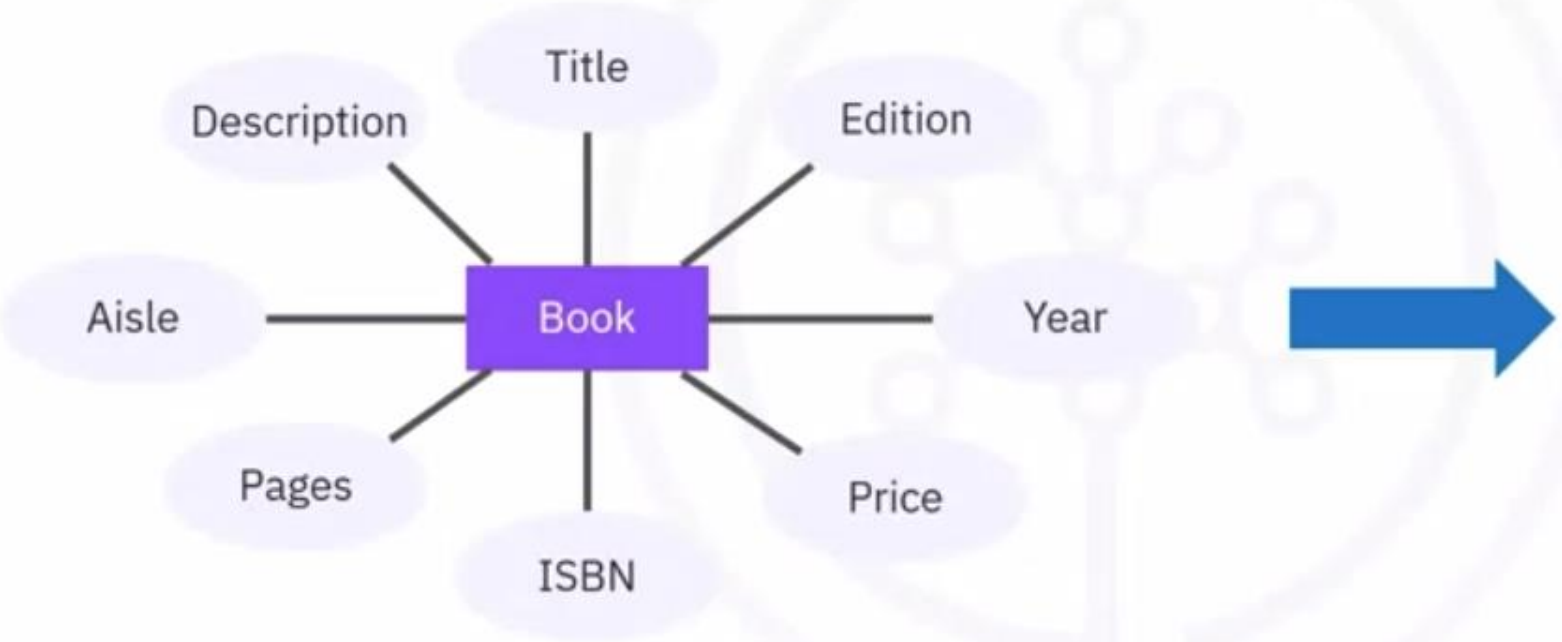
- Most used data model
- Allows for data independence
- Data is stored in tables



logical data independence - physical data independence - physical storage independence

Entity-Relationship Model

- Used as a tool to design relational databases



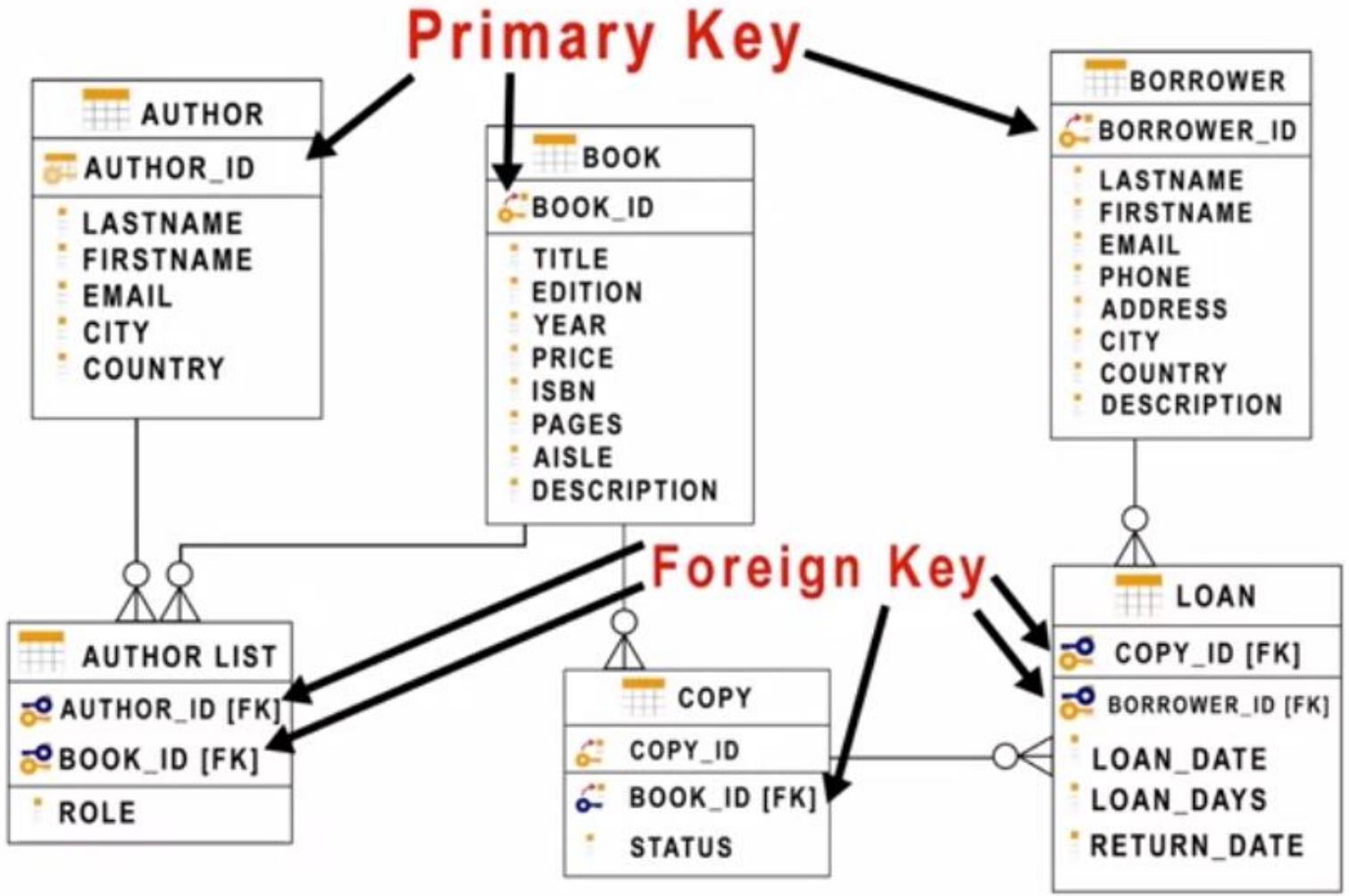
Mapping Entity Diagrams to Tables

- Entities become tables
- Attributes get translated into columns

Table: Book

Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-98662 83-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866 283-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C, the free version of DB2

Primary Keys and Foreign Keys



Summary

Now you know:

- The key advantage of the relational model is data independence
- Entities are independent objects which have Attributes
- Entities map to Tables in a Relational Database
- Attributes map to Columns in a Table
- Common data types include characters, numbers, and dates/times
- A Primary Key uniquely identifies a specific row in a table

Types of SQL Statements - DDL

- SQL Statement types: DDL and DML

- DDL (Data Definition Language) statements:
  - Define, change, or drop data

- Common DDL:
  - CREATE
  - ALTER
  - TRUNCATE
  - DROP

Types of SQL Statements - DML

- DML (Data Manipulation Language) statements:
  - Read and modify data
  - CRUD operations (Create, Read, Update & Delete rows)

- Common DML:
  - INSERT
  - SELECT
  - UPDATE
  - DELETE

CREATE table

- **Syntax:**

```
CREATE TABLE table_name
   (
    column_name_1 datatype optional_parameters,
    column_name_2 datatype,
    …
    column_name_n datatype
   )
```

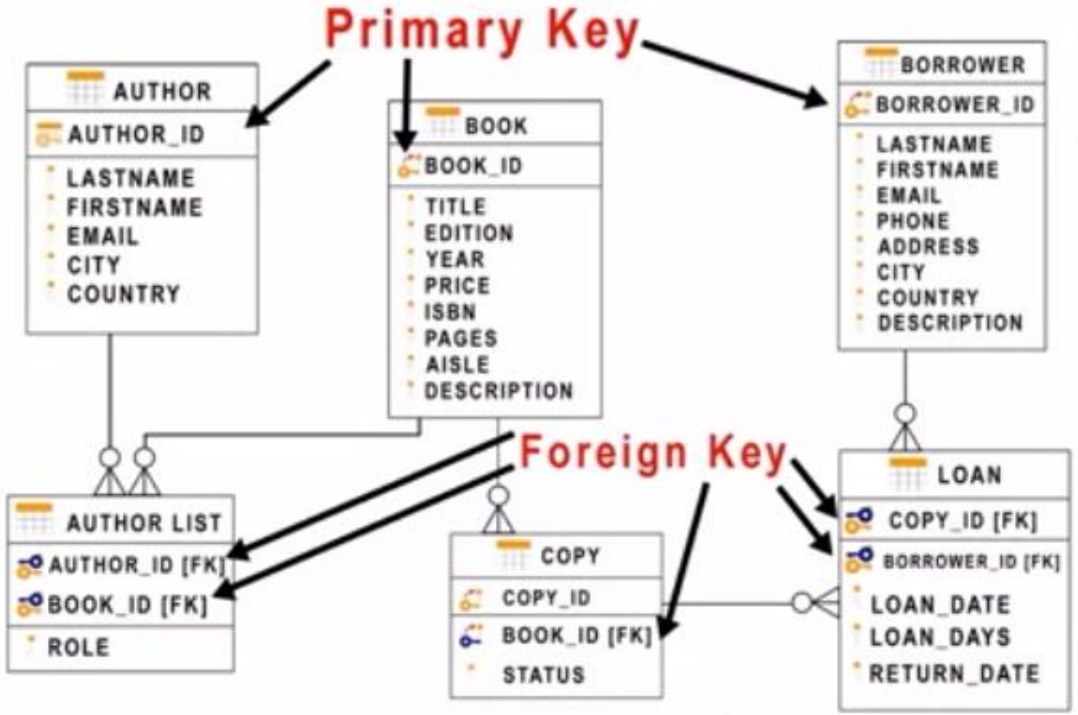
EXAMPLE

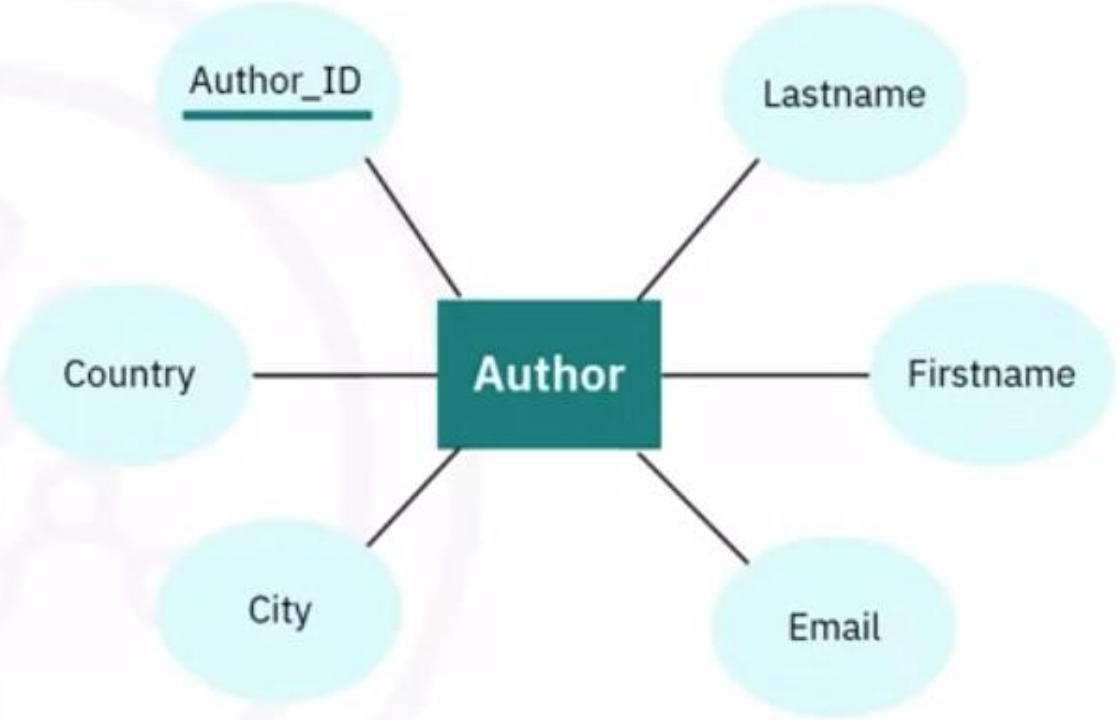
- Create a table for Canadian provinces

```
CREATE TABLE provinces(
    id char(2) PRIMARY KEY NOT NULL,
    name varchar(24)
   )
```

id *char(2)*	**name** *varchar(24)*
AB	ALBERTA
BC	BRITISH COLUMBIA
...	...

Create a table





Primary Key: Uniquely Identifies each Row in a Table

CREATE TABLE Statement

To create the Author table, use the following columns and datatypes:

AUTHOR(Author_ID:char, Lastname:varchar, Firstname:varchar, Email:varchar, City:varchar, Country:char)

```
CREATE TABLE author (
    author_id CHAR(2) PRIMARY KEY NOT NULL,
    lastname VARCHAR(15) NOT NULL,
    firstname VARCHAR(15) NOT NULL,
    email VARCHAR(40),
    city VARCHAR(15),
    country CHAR(2)
   )
```

ALTER TABLE ... ADD COLUMN

- Add or remove columns
- Modify the data type of columns
- Add or remove keys
- Add or remove constraints

```
ALTER TABLE <table_name>
  ADD COLUMN <column_name_1> datatype
  . . .
  ADD COLUMN <column_name_n> datatype;
```

ALTER TABLE ... ALTER COLUMN

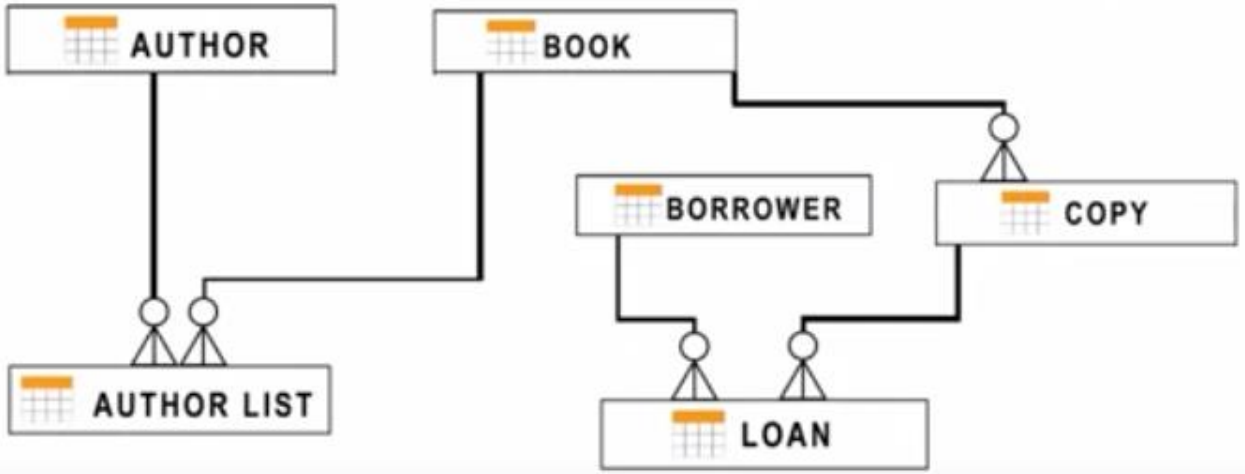
```
ALTER TABLE author
  ALTER COLUMN telephone_number SET DATA TYPE
CHAR(20);
```

author_id	lastname	firstname	email	city	country	telephone_number
1001	Thomas	John	johnt@...	New York	USA	555-1111
1002	James	Alice	alicej@...	Seattle	USA	555-1112
1003	Wells	Steve	stevew@...	Montreal	Canada	555-2222
1004	Kumar	Santosh	kumars@...	London	UK	555-3333

DROP TABLE

```
DROP TABLE <table_name>;
```

```
DROP TABLE author;
```



TRUNCATE TABLE

```
TRUNCATE TABLE author
    IMMEDIATE;
```

author_id	lastname	firstname	email	city	country	
1001	Thomas	John	johnt@...	New York	USA	
1002	James	Alice	alicej@...	Seattle	USA	
1003	Wells	Steve	stevew:@...	Montreal	Canada	
1004	Kumar	Santosh	kumars@...	London	UK	

1004	Kumar	Santosh	kumars@...	London	UK	555-9999

TRUNCATE Table

TRUNCATE TABLE statements are used to delete all of the rows in a table. The syntax of the statement is:

```
TRUNCATE TABLE table_name;
```

So, to truncate the "author" table, the statement will be written as:

```
TRUNCATE TABLE author;
```

The output would be as shown in the image below.

author_id	lastname	firstname	email	city	country

Note: *The TRUNCATE statement will delete the rows and not the table.*

Author

Retrieving rows from a table

```
db2 => select * from Book
```

Book_ID	Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
B1	Getting started with DB2 Express-C	1	2010	24.99	978-0-98006283-1-1	300	DB-A02	Teaches you the fundamentals
B2	Database Fundamentals	1	2010	24.99	978-0-98666283-5-1	280	DB-A01	Teaches you the essentials of
B3	Getting started with DB2 App Dev	1	2011	35.99	978-0-98086283-4-1	345	DB-A03	Teaches you the essentials of
B4	Getting started with WAS CE	1	2010	49.99	978-0-98946283-3-1	458	DB-A04	Teaches you the essentials of

4 record(s) selected.

```
db2 => select book_id, title from Book
```

Book_ID	Title
B1	Getting started with DB2 Express-C
B2	Database Fundamentals
B3	Getting started with DB2 App Dev
B4	Getting started with WAS CE

4 record(s) selected.

```
db2 => select book_id, title from Book
       WHERE book_id='B1'
```

Book_ID	Title
B1	Getting started with DB2 Express-C

1 record(s) selected.

SELECT star from Book gives a result set of four rows.

Retrieving rows - using a String Pattern

```
db2 => select firstname from Author
         WHERE firstname like 'R%'

Firstname
---------------------------------------------------------
Raul
Rav
    2 record(s) selected.
```

The like predicate is used in a WHERE clause to search for a pattern in a column.

Retrieving rows - using a Range

```
db2 => select title, pages from Book
           WHERE pages >= 290 AND pages <= 300
```

Title	Pages
Database Fundamentals	300
Getting started with DB2 App Dev	298

2 record(s) selected.

```
db2 => select title, pages from Book
           WHERE pages between 290 and 300
```

Title	Pages
Database Fundamentals	300
Getting started with DB2 App Dev	298

2 record(s) selected.

but the SELECT statement is easier and quicker to write.

Retrieving rows - using a Set of Values

```
db2 => select firstname, lastname, country from Author
          WHERE country='AU' OR country='BR'
```

Firstname	Lastname	Country
Xiqiang	Ji	AU
Juliano	Martins	BR

2 record(s) selected.

```
db2 => select firstname, lastname, country from Author
          WHERE country IN ('AU','BR')
```

Firstname	Lastname	Country
Xiqiang	Ji	AU
Juliano	Martins	BR

In this case the countries Australia or Brazil.

Using the ORDER BY clause

```
db2 => select title from Book

Title
----------------------------------------------------------
Getting started with DB2 Express-C
Database Fundamentals
Getting started with DB2 App Dev
Getting started with WAS CE
                4 record(s) selected.
```

```
db2 => select title from Book
       ORDER BY title

Title
----------------------------------------------------------
Database Fundamentals
Getting started with DB2 App Dev
Getting started with DB2 Express-C
Getting started with WAS CE
    4 record(s) selected.
```

By default the result set is sorted in ascending order

In this example, we have used order by on the column title to sort the result set.

ORDER BY clause – Descending order

```
db2 => select title from Book
       ORDER BY title

Title
----------------------------------------------------------
Database Fundamentals
Getting started with DB2 App Dev
Getting started with DB2 Express-C
Getting started with WAS CE
    4 record(s) selected.
```

Ascending order by default

```
db2 => select title from Book
       ORDER BY title DESC

Title
----------------------------------------------------------
Getting started with WAS CE
Getting started with DB2 Express-C
Getting started with App Dev
Database Fundamentals
    4 record(s) selected.
```

Descending order with DESC keyword

The result set is now sorted according to the column specified,

Specifying Column Sequence Number

```
db2 => select title, pages from Book
          ORDER BY 2
```

Title	Pages
Getting started with WAS CE	278
Getting started with DB2 Express-C	280
Getting started with App Dev	298
Database Fundamentals	300

4 record(s) selected.

Ascending order by Column 2 (number of pages)

so the sort order is based on the values in the pages column.

Eliminating Duplicates - DISTINCT clause

```
db2 => select country from Author
       ORDER BY 1
```

Country
AU
BR
...
CN
CN
...
IN
IN
IN
...
RO
RO

20 record(s) selected.

```
db2 => select distinct(country)
       from Author
```

Country
AU
BR
CA
CN
IN
RO

6 record(s) selected.

To eliminate duplicates, we use the keyword distinct.

GROUP BY clause

```
db2 => select country from Author
       ORDER BY 1
```

Country
AU
BR
...
CN
CN
...
IN
IN
IN
...
RO
RO

20 record(s) selected.

```
db2 => select country, count(country)
       from Author GROUP BY country
```

Country	2
AU	1
BR	1
CA	3
CN	6
IN	6
RO	3

6 record(s) selected.

Instead of using the column named "2,"

GROUP BY clause

```
db2 => select country from Author
       ORDER BY 1
```

Country
AU
BR
...
CN
CN
...
IN
IN
IN
...
RO
RO

20 rec

```
db2 => select country, count(country)
       as Count from Author group by country
```

Country	Count
AU	1
BR	1
CA	3
CN	6
IN	6
RO	3

6 record(s) selected.

In this example, we change the derived column name

Restricting the Result Set - HAVING clause

```
db2 => select country, count(country)
         as Count from Author group by country
```

Country	Count
AU	1
BR	1
CA	3
CN	6
IN	6
RO	3

6 record(s) selected.

```
db2 => select country, count(country)
    as Count from Author
    group by country
    having count(country) > 4
```

Country	Count
CN	6
IN	6

6 record(s) selected.

In this example, those countries are China with six authors and India,

Built-in Functions

- Most databases come with built-in SQL functions
- Built-in functions can be included as part of SQL statements
- Database functions can significantly reduce the amount of data that needs to be retrieved
- Can speed up data processing

that is, user-defined functions, in the database;
but that is a more advanced topic.

PETRESCUE TABLE

PETRESCUE

ID INTEGER	ANIMAL VARCHAR(20)	QUANTITY INTEGER	COST DECIMAL(6,2)	RESCUEDATE DATE
1	Cat	9	450.09	2018-05-29
2	Dog	3	666.66	2018-06-01
3	Dog	1	100.00	2018-06-04
4	Parrot	2	50.00	2018-06-04
5	Dog	1	75.75	2018-06-10
6	Hamster	6	60.60	2018-06-11
7	Cat	1	44.44	2018-06-11
8	Goldfish	24	48.48	2018-06-14
9	Dog	2	222.22	2018-06-15

Aggregate or Column Functions

- INPUT: Collection of values (e.g. entire column)
- Output: Single value
- Examples: SUM(), MIN(), MAX(), AVG(), etc.

Examples of aggregate functions include: sum, minimum, maximum, average, etc.

SUM

SUM function: Add up all the values in a column

```
SUM(COLUMN_NAME)
```

Example 1: Add all values in the COST column:

```
select SUM(COST) from PETRESCUE
```

Example 1: Result:

```
1
1718.24
```

When you use an aggregate function,
the column in the result set

Column Alias

Example 2: Explicitly name the output column SUM_OF_COST:

```
select SUM(COST) as SUM_OF_COST
       from PETRESCUE
```

Example 2: Results:

```
SUM_OF_COST
1718.24
```

Note the use of 'as' in this example.

MIN, MAX

MIN: Return the MINIMUM value
MAX: Return the MAXIMUM value

Example 3A. Get the maximum QUANTITY of any ANIMAL:

```
select MAX(QUANTITY) from PETRESCUE
```

Example 3B. Results:

```
1
24
```

a single transaction,
select MAX(QUANTITY) from PETRESCUE.

MIN, MAX

Example 3B. Get the minimum value of ID column for Dogs:

```
select MIN(ID) from PETRESCUE where ANIMAL =  'Dog'
```

Example 3B. Results:

```
1
2
```

select MIN(ID) from PETRESCUE where animal equals dog.

Average

AVG() return the average value

Example 4. Specify the Average value of COST:

```
select AVG(COST) from PETRESCUE
```

Example 4. Results:

```
1
190.91555555555555555555555555
```

as: select AVG(COST) from PETRESCUE.

Average

Mathematical operations can be performed between columns.

Example 5. Calculate the average COST per ‘Dog’:

```
select AVG(COST / QUANTITY) from PETRESCUE
where ANIMAL =  'Dog'
```

Example 5. Results:

```
1
127.270000000000000000000000000
```

In this case, the cost is for multiple units;
so we first divide

SCALAR and STRING FUNCTIONS

SCALAR: Perform operations on every input value
Examples: ROUND(), LENGTH(), UCASE, LCASE

Example 6: Round UP or DOWN every value in COST:

```
select
      ROUND(COST)
      from PETRESCUE
```

Example 6. Results:

```
1
450.00
667.00
100.00
50.00
76.00
```

SCALAR and STRING FUNCTIONS

SCALAR: Perform operations on every input value
Examples: ROUND(), LENGTH(), UCASE, LCASE

Example 7. Retrieve the length of each value in ANIMAL:

```
select
      LENGTH(ANIMAL)
      from PETRESCUE
```

Example 7. Results:

```
1
3
3
3
6
3
```

UCASE, LCASE

Example 8: Retrieve ANIMAL values in UPPERCASE:

```
select UCASE(ANIMAL) from PETRESCUE
```

Example 8: Results:

```
1
CAT
DOG
DOG
PARROT
DOG
```

UCASE, LCASE

Example 9: Use the function in a WHERE clause :

```
select * from PETRESCUE
where LCASE(ANIMAL) = 'cat'
```

Example 9: Results:

ID	ANIMAL	QUANTITY	COST	DATE
1	Cat	9	450.09	2018-05-29
7	Cat	1	44.44	2018-06-11

UCASE, LCASE

Example 10: Use the DISTINCT() function to get unique values :

```
select DISTINCT(UCASE(ANIMAL)) from PETRESCUE
```

Example 10: Results:

```
1
CAT
DOG
GOLDFISH
HAMSTER
PARROT
```

Date, Time Functions

Most databases contain special datatypes for dates and times

DATE: `YYYYMMDD`

TIME: `HHMMSS`

TIMESTAMP: `YYYYXXDDHHMMSSZZZZZZ`

Date / Time functions:

```
YEAR(), MONTH(), DAY(), DAYOFMONTH(), DAYOFWEEK(),
DAYOFYEAR(), WEEK(), HOUR(), MINUTE(), SECOND()
```

Date, Time Functions (continued)

Example 11: Extract the DAY portion from a date:

```
select DAY(RESCUEDATE) from PETRESCUE
where ANIMAL='Cat'
```

Example 11: Results:

ID	ANIMAL	QUANTITY	COST	RESCUEDATE
1	Cat	9	450.09	2018-05-29
7	Cat	1	44.44	2018-06-11

29

11

Date, Time Functions (continued)

Example 12: Get the number of rescues during the month of May:

```
select COUNT(*) from PETRESCUE
where MONTH(RESCUEDATE)='05'
```

Example 12: Results:

```
1
```

ID	ANIMAL	QUANTITY	COST	RESCUEDATE
1	Cat	9	450.09	2018-05-29
7	Cat	1	44.44	2018-06-11

Date or Time Arithmetic

Example 13: What date is it 3 days after each rescue date?

```
Select DATE_ADD(RESCUEDATE, INTERVAL 3 DAY) from PETRESCUE
```

Example 13: Results:

	ID	ANIMAL	QUANTITY	COST	RESCUEDATE	+ 3 DAYS
2018-06-01	1	Cat	9	450.09	2018-05-29	2018-06-01
2018-06-04	2	Dog	3	666.66	2018-06-01	2018-06-04
2018-06-07	3	Dog	1	100	2018-06-04	2018-06-07
2018-06-07	4	Parrot	2	50	2018-06-04	2018-06-07
2018-06-13	5	Dog	1	75.75	2018-06-10	2018-06-13

Date or Time Arithmetic

Special Registers:

```
CURRENT_DATE, CURRENT_TIME
```

Example 14: Find how many days have passed since each RESCUEDATE till now:

```
Select FROM_DAYS(DATEDIFF(CURRENT_DATE, RESCUEDATE)) from PETRESCUE
```

Example 14: Sample result (format YYYY-MM-DD):

```
0001-09-21
```

Summary

Now you know that:

- SQL contains DATE, TIME, and TIMESTAMP types.
- DATE has 8 digits: YYYYMMDD
- TIME has six digits: HHMMSS
- TIMESTAMP has 20 digits: YYYYXXDDHHMMSSZZZZZZ where XX represents month and ZZZZZZ represents microseconds.
- Functions exist to extract the DAY, MONTH, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, WEEK, HOUR, MINUTE, SECOND.
- Date and Time functions can be used in the WHERE clause.
- The DAY function can be used to Extract the DAY portion from a date.

Sub-queries and Nested Selects

Sub-query: A query inside another query

```
select COLUMN1 from TABLE
  where COLUMN2 = (select MAX(COLUMN2) from TABLE)
```

EMPLOYEES

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000	30002	5

Why use sub-queries?

To retrieve the list of employees who earn more than the average salary:

```
select * from employees
        where salary > AVG(salary)
```

This query will result in error:

```
SQL0120N Invalid use of an aggregate function or OLAP
function.SQLCODE=-120, SQLSTATE=42903
```

Sub-queries to evaluate Aggregate functions

- Cannot evaluate Aggregate functions like AVG() in the WHERE clause –
- Therefore, use a sub-Select expression:

```
select EMP_ID, F_NAME, L_NAME, SALARY
       from employees
       where SALARY <
       (select AVG(SALARY) from employees);
```

Sub-queries to evaluate Aggregate functions

Result:

EMP_ID	F_NAME	L_NAME	SALARY
E1003	Steve	Wells	50000.00
E1004	Santosh	Kumar	60000.00
E1007	Mary	Thomas	65000.00

Sub-queries in list of columns

- Substitute column name with a sub-query
- Called Column Expressions

```
select EMP_ID, SALARY, AVG(SALARY) AS AVG_SALARY
       from employees ;
```

```
select EMP_ID, SALARY,
       ( select AVG(SALARY) from employees )
                AS AVG_SALARY
       from employees ;
```

Sub-queries in list of columns

Result:

EMP_ID	SALARY	AVG_SALARY
E1002	80000.00	68888.888888888888888888888
E1003	50000.00	68888.888888888888888888888
E1004	60000.00	68888.888888888888888888888
E1005	70000.00	68888.888888888888888888888
E1006	90000.00	68888.888888888888888888888
E1007	65000.00	68888.888888888888888888888
E1008	65000.00	68888.888888888888888888888
E1009	70000.00	68888.888888888888888888888
E1010	70000.00	68888.888888888888888888888

Sub-queries in FROM clause

- Substitute the TABLE name with a sub-query
- Called Derived Tables or Table Expressions
- Example:

```
select * from
        ( select EMP_ID, F_NAME, L_NAME, DEP_ID
                from employees) AS EMP4ALL ;
```

such as when working with
multiple tables and doing

Sub-queries in FROM clause

Result:

EMP_ID	F_NAME	L_NAME	DEP_ID
E1002	Alice	James	5
E1003	Steve	Wells	5
E1004	Santosh	Kumar	5
E1005	Ahmed	Hussain	2
E1006	Nancy	Allen	2
E1007	Mary	Thomas	7
E1008	Bharath	Gupta	7
E1009	Andrea	Jones	7
E1010	Ann	Jacob	5

joins.

Working with multiple tables

Ways to access multiple tables in the same query:

1. Sub-queries
2. Implicit JOIN
3. JOIN operators (INNER JOIN, OUTER JOIN, and so on)

The third option is covered in more detail in other videos.

Accessing multiple tables with sub-queries

EMPLOYEES:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000	30002	5

DEPARTMENTS:

DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
5	Software Development	30002	L0002
7	Des		

ID exists in the departments table, we can use a sub-query as follows.

Accessing multiple tables with sub-queries

Query:

```
select * from employees where DEP_ID IN
       ( select DEPT_ID_DEP from departments );
```

Result:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1002	Alice	James	123457	7/31/1972	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	8/10/1980	M	291 Springs, Gary,IL	300	50000	30002	5
E1004	Santosh	Kumar	123459	7/20/1985	M	511 Aurora Av, Aurora,IL	400	60000	30004	5
E1[illegible]	Mary	Thomas	[illegible]	5/5/1975	F	100 Rose Pl, G[illegible]	650	[illegible]	30003	7
E1009	Andrea	Jones	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]03	7
E1010	Ann	Jacob	123415	3/30/1982	F	[illegible] Springs,Elgin,IL	220	70000	30004	5

table is used for filtering the result set of the outer query.

Multiple tables with sub-queries

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000	30002	5

DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
5	Software Development	30002	L0002
7	Design Team	30003	L0003

Therefore, we can use a sub-query from the Departments table as input to the employee

Multiple tables with sub-queries

Query:

```
select * from employees where DEP_ID IN
       ( select DEPT_ID_DEP from departments where LOC_ID = 'L0002' );
```

Result:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1002	Alice	James	123457	7/31/1972	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	8/10/1980	M	291 Springs, Gary,IL	300	50000	30002	5
E1004	Santosh	Kumar	123459	7/20/1985	M	511 Aurora Av, Aurora,IL	400	60000	30004	5
E1010	Ann	Ja		3/30/198		111 Britany			30004	5

Now, let's retrieve the department ID and department name for employees who earn more

Multiple tables with sub-queries

Query:

```
select DEPT_ID_DEP, DEP_NAME from departments
 where DEPT_ID_DEP IN
  ( select DEP_ID from employees where SALARY > 70000 ) ;
```

Result:

DEPT_ID_DEP	DEP_NAME
5	Software Group

IN, select department_ID from employees where salary is greater than 70,000.

Accessing multiple tables with Implicit Join

Specify 2 tables in the FROM clause:

```
select * from employees, departments;
```

The result is a full join (or Cartesian join):

- Every row in the first table is joined with every row in the second table
- The result set will have more rows than in both tables

If you examine the results set, you will see more rows than in both tables individually.

Accessing multiple tables with Implicit Join

Use additional operands to limit the result set:

```
select * from employees, departments
       where employees.DEP_ID =
             departments.DEPT_ID_DEP;
```

Use shorter aliases for table names:

```
select * from employees E, departments D
       where E.DEP_ID = D.DEPT_ID_DEP;
```

use these aliases in the WHERE clause.

Accessing multiple tables with Implicit Join

Query:

```
select * from employees E, departments D
        where E.DEP_ID = D.DEPT_ID_DEP;
```

Result:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID	DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
E1002	Alice	James	123457	7/31/1972	F	980 Berry ln, Elgin,IL	200	80000	30002	5	5	Software Group	30002	L0002
E1003	Steve	Wells	123458	8/10/1980	M	291 Springs, Gary,IL	300	50000	30002	5	5	Software Group	30002	L0002
E1004	Santosh	Kumar	123459	7/20/1985	M	511 Aurora Av, Aurora,IL	400	60000	30002	5	5	Software Group	30002	L0002
E1007	Mary	Thomas	123412	5/5/1975	F	100 Rose Pl, Gary,IL	650	65000	30003	7	7	Design Team	30003	L0003
E1008	Bharath	Gupta	123413	5/6/1985	M	145 Berry Ln, Naperville,IL	660	65000	30003	7	7	Design Team	30003	L0003
E1009	Andrea	Jones	123414	7/9/1990	F	120 Fall Creek, Gary,IL	234	70000	30003	7	7	Design Team	30003	L0003
E1010	Ann	Jacob	123415	8/30/1982	F	111 Britany [illegible]	220	70000	30002	5	5	[illegible]re Group	30002	L0002

use these aliases in the WHERE clause.

Accessing multiple tables with Implicit Join

To see the department name for each employee:

```
select EMP_ID, DEP_NAME
       from employees E, departments D
       where E.DEP_ID = D.DEPT_ID_DEP;
```

Column names in the select clause can be pre-fixed by aliases:

```
select E.EMP_ID, D.DEP_ID_DEP from
       employees E, departments D
       where E.DEP_ID = D.DEPT_ID_DEP
```

as shown in the query.

Accessing multiple tables with Implicit Join

Query:

```
select EMP_ID, DEP_NAME from employees E, departments D
      where E.DEP_ID = D.DEPT_ID_DEP;
```

Result:

EMP_ID	DEP_NAME
E1002	Software Group
E1003	Software Group
E1004	Software Group
E1007	Design Team
E1008	Design Team
E1009	
E1010	

In this lesson, we have shown you how to work with multiple tables using sub-queries and

To exit full screen, press Esc

How to Access Databases Using Python

Hello. In this video you will learn how to access databases using Python.

0:01 / 6:00

Benefits of python for database programming

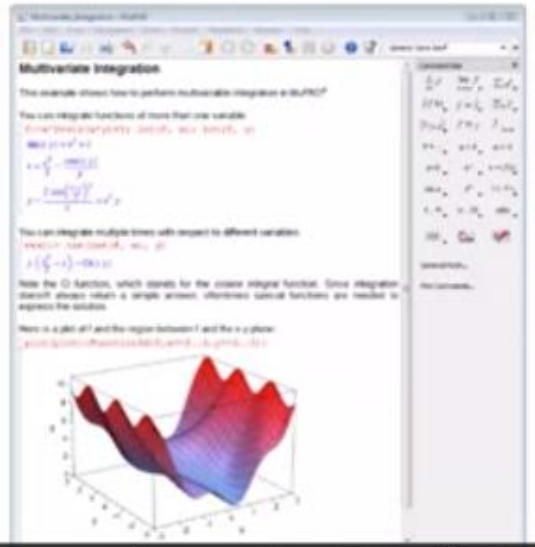
- Python ecosystem: NumPy, pandas, matplotlib, SciPy
- Ease of use
- Portable
- Python supports relational database systems
- Python Database API (DB-API)
- Python detailed documentation

Notebooks are also very popular in the field of data science because they run

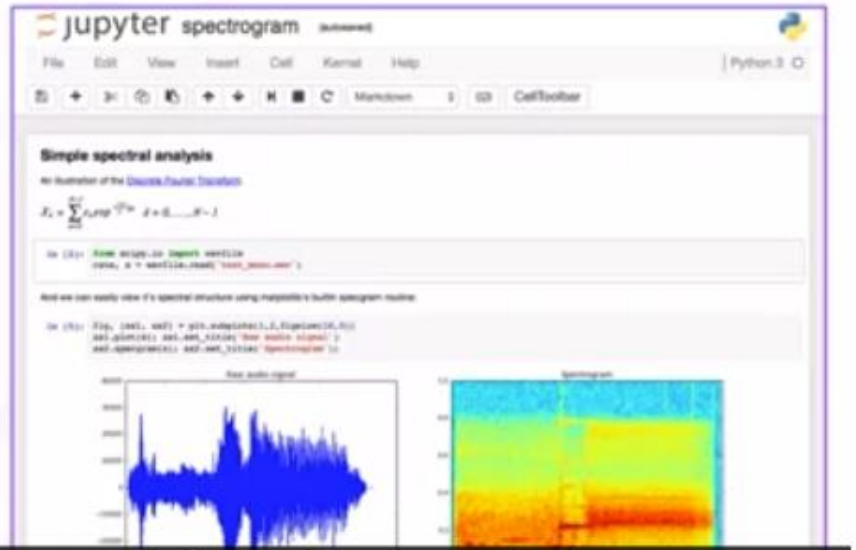
Introduction to notebooks

Notebooks allows creating and sharing documents containing live codes, equation, visualizations, and explanatory text.

Matlab notebook



Jupyter notebook



Apache Zeppelin, Apache Spark notebook, and the Databricks cloud.

What are Jupyter Notebooks?

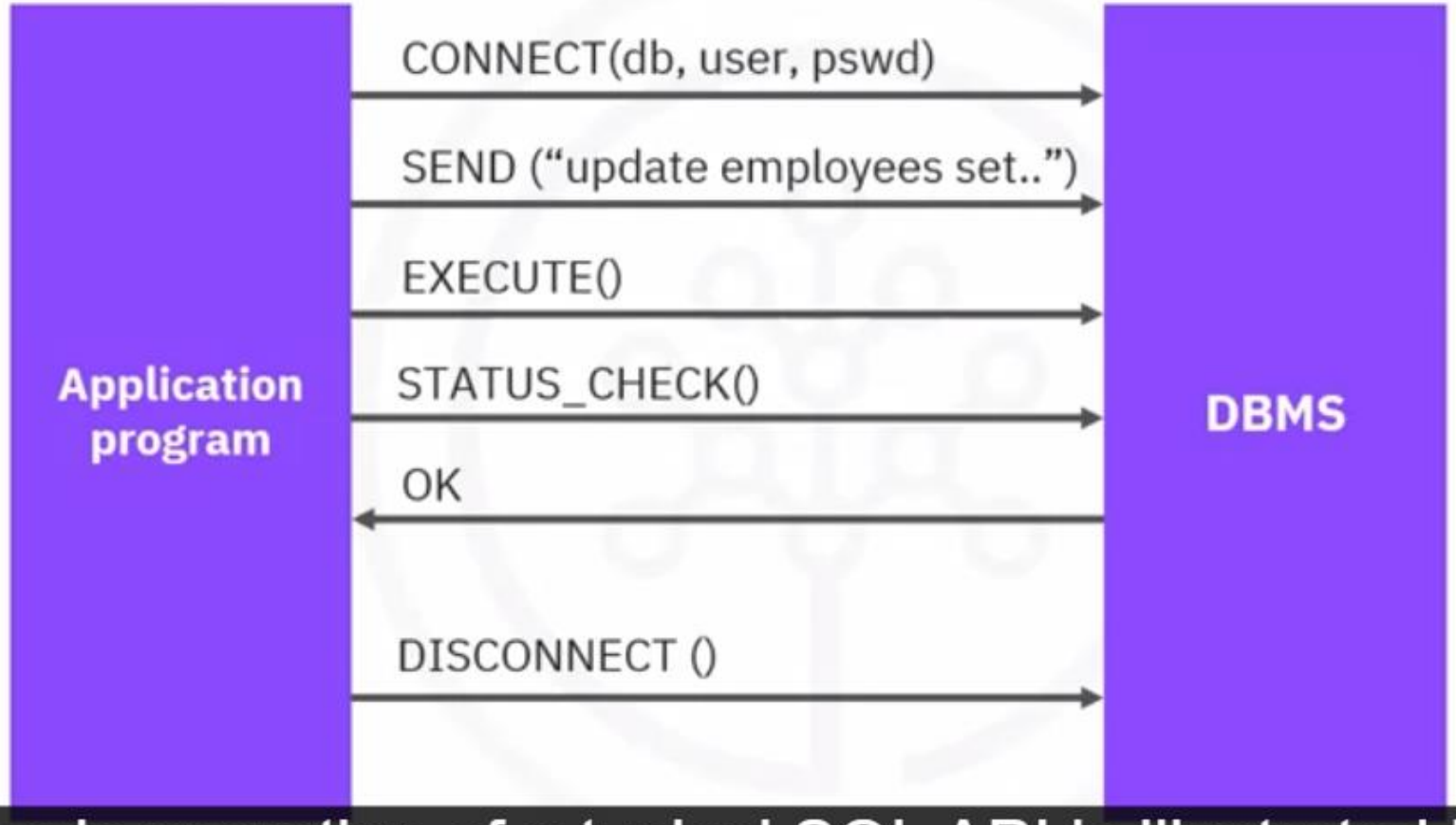
- Language of choice
- Share notebooks
- Interactive output
- Big data integration

scikit-learn, ggplot2, and TensorFlow.

Accessing databases using Python

We will explain the basics of SQL APIs and Python DB APIs.

What is a SQL API?



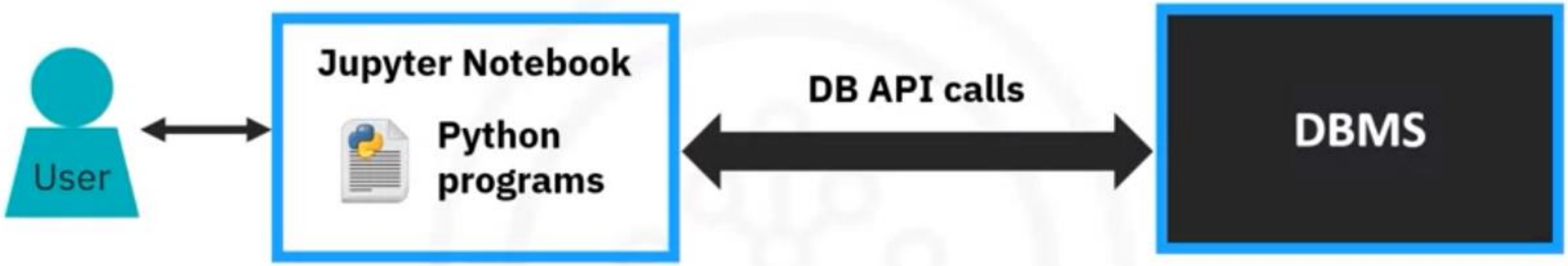
The basic operation of a typical SQL API is illustrated in the figure.

APIs used by popular SQL-based DBMS systems

Application or Database	SQL API
MySQL	MySQL C API
PostgreSQL	psycopg2
IBM DB2	ibm_db
SQL Server	dblib API
Database access for Microsoft Windows OS	ODBC
Oracle	OCI
Java	JDBC

And finally, JDBC is used by Java applications.

What is a DB-API?



- Python's standard API for accessing relational databases
- Allows a single program that to work with multiple kinds of relational databases
- Learn DB-API functions once, use them with any database

Benefits of using DB-API

- Easy to implement and understand
- Encourages similarity between the Python modules used to access databases
- Achieves consistency
- Portable across databases
- Broad reach of database connectivity from Python

Examples of libraries used by database systems to connect to Python applications

Database	DB API
DB2 Warehouse on Cloud	Ibm_db
Compose for MySQL	MySQL Connector/Python
Compose for PostgreSQL	psycopg2
Compose for MongoDB	PyMongo

Concepts of the Python DB API

Connection Objects

- Database connections
- Manage transactions

Cursor Objects

- Database Queries
- Scroll through result set
- Retrieve results

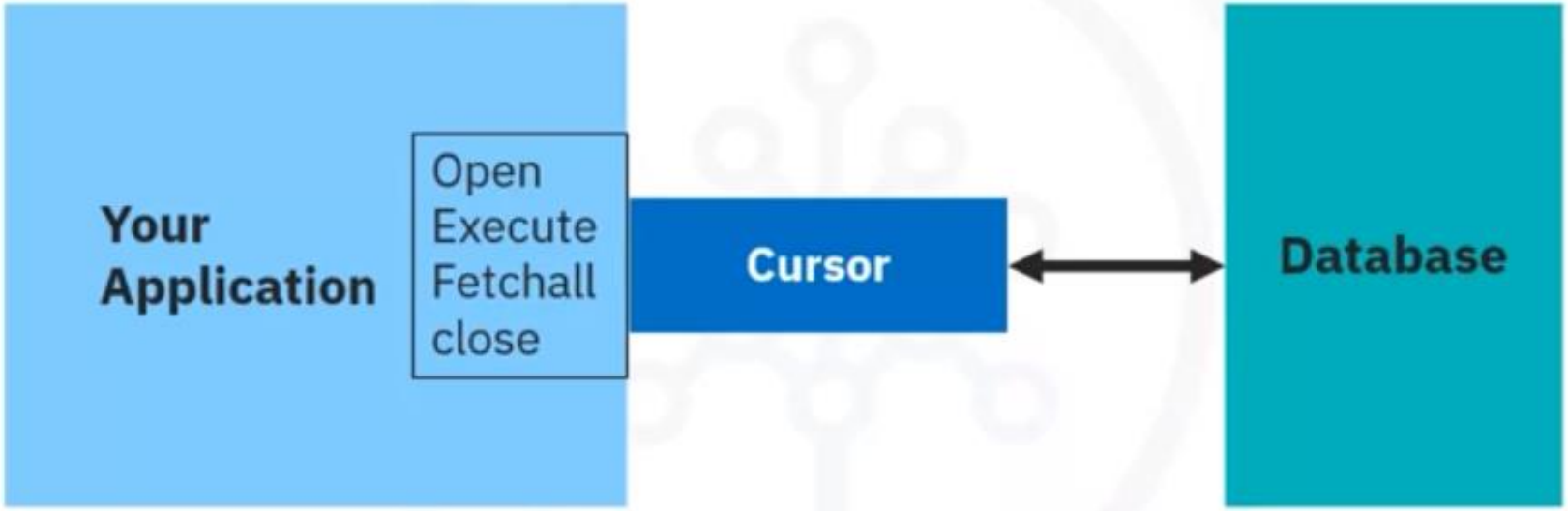
What are Connection methods?

- .cursor()
- .commit()
- .rollback()
- .close()

What are cursor methods?

- .callproc()
- .execute()
- .executemany()
- .fetchone()
- .fetchmany()
- .fetchall()
- .nextset()
- .arraysize()
- .close()

What is a database cursor?



the program's current position within the query results.

Writing code using DB-API

```
from dbmodule import connect
#Create connection object
Connection =
connect('databasename',
'username', 'pswd')

#Create a cursor object
Cursor=connection.cursor()
```

```
#Run Queries
Cursor.execute('select * from
mytable')
Results=cursor.fetchall()

#Free resources
Cursor.close()
Connection.close()
```

to avoid unused connections
taking up resources.

Magic Statements in Jupyter Notebooks

Magic commands are special commands that provide special functionalities.

- They are not valid Python code but affect the behavior of the notebook.
- They are designed to solve standard data analysis problems.

in standard data analysis.

Types of Cell Magics

Commands that are prefixed with a single % character and operate on a single line of input.

Commands that are prefixed with two %% characters and operate on multiple lines of input.

execute the cell in a different programming language.

Using Line Magic Statements

Line Magics	Uses
%pwd	prints the current working directory
%ls	lists all files in the current directory
%history	shows the command history
%reset	resets the namespace by removing all names defined by the user
%who	lists all variables in the namespace
%whos	provides more detailed information about all variables in the namespace
%matplotlib inline	makes matplotlib plots appear within the notebook
%timeit	times the execution of a single statement
%lsmagic	lists all available line magics

can use the percentage
LS magic command.

Using Line Magic Statements

```
%pwd
%ls
```

Both Line magics in the same cell

```
%timeit <statement>
```

Line Magic: Time for executing single statement

```
%%timeit
<statement_1>
<statement_2>
<statement_3>
```

Cell Magic: Time for executing the whole cell

```
%%writefile myfile.txt
<statement_1>
<statement_2>
<statement_3>
```

Writes all statements of the cell to myfile.txt

Using Cell Magic Statements

%%HTML	Write HTML code in cells and render it

```
%%HTML
<h>Hello world</h1>
```

%%javascript	Write JavaScript code in cells

%%bash cell	Write bash commands

```
%%bash
echo “Hello world!”
Hello, World!
```

Using SQL Magic

- Install ipython-sql by running the following statement:

```
!pip install --user ipython-sql
```

- Enable the SQL magic in Juypter notebook using this statement:

```
%load_ext sql
```

Using SQL Magic with SQLite Database

```
import sqlite3
conn = sqlite3.connection('HR.db')
```

```
%load_ext sql
```

```
%sql sqlite:///HR.db
```

```
%sql SELECT * FROM Employee
```

of employee table
in the HR database.

Load CSV to SQLite3 with Pandas

```
import pandas as pd
import sqlite3
data = pd.read_csv('./menu.csv')
conn = sqlite3.connect('McDonalds.db')
data.to_sql('MCDONALDS_NUTRITION', conn)
```

as a table MCDONALDS_NUTRITION, which can now be queried as per requirement.

Using pandas

```
df = pd.read_sql("SELECT * FROM MCDONALDS_NUTRITION", conn)
print(df)
```

Out[21]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	Di Fit (% Da Va
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4	17
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4	17
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4	17
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4	17
4	Breakfast	Sausage McMuffin with Egg Whites	5.7 oz (161 g)	400	210	23.0	35	8.0	42	0.0	...	30	10	4	17
5	Breakfast	Steak & Egg McMuffin	6.5 oz (185 g	430	210	23.0	36	9.0	46	1.0	...	31	10	4	18

are passed as parameters to the read_sqlmethod.

View first few rows

```
df1.head()
```

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	Dietary Fiber (% Daily Value)	S
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4	17	3
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4	17	3
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4	17	2
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4	17	2
4	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	210	23.0	35	8.0	42	0.0	...	30	10	4	17	2

that we created using the head method.

Learn about your data

```
df.describe(include = 'all')
```

Out[34]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Ca (% Va
count	260	260	260	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	...	260.000000	26
unique	9	260	107	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
top	Coffee & Tea	Nonfat Caramel Mocha (Large)	16 fl oz cup	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
freq	95	1	45	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
mean	NaN	NaN	NaN	368.269231	127.096154	14.165385	21.815385	6.007692	29.965385	0.203846	...	47.346154	15
std	NaN	NaN	NaN	240.269886	127.875914	14.205998	21.885199	5.321873	26.639209	0.429133	...	28.252232	9.4
min	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0
25%	NaN	NaN	NaN	210.000000	20.000000	2.375000	3.750000	1.000000	4.750000	0.000000	...	30.000000	10
50%	NaN	NaN	NaN	340.000000	100.000000	11.000000	17.000000	5.000000	24.000000	0.000000	...	44.000000	15
75%	NaN	NaN	NaN	500.000000	200.000000	22.250000	35.000000	10.000000	48.000000	0.000000	...	60.000000	20
max	NaN	NaN	NaN	1880.000000	10				000000	2.500000	...	141.000000	47

statistical methods.

Which food item has maximum sodium content?

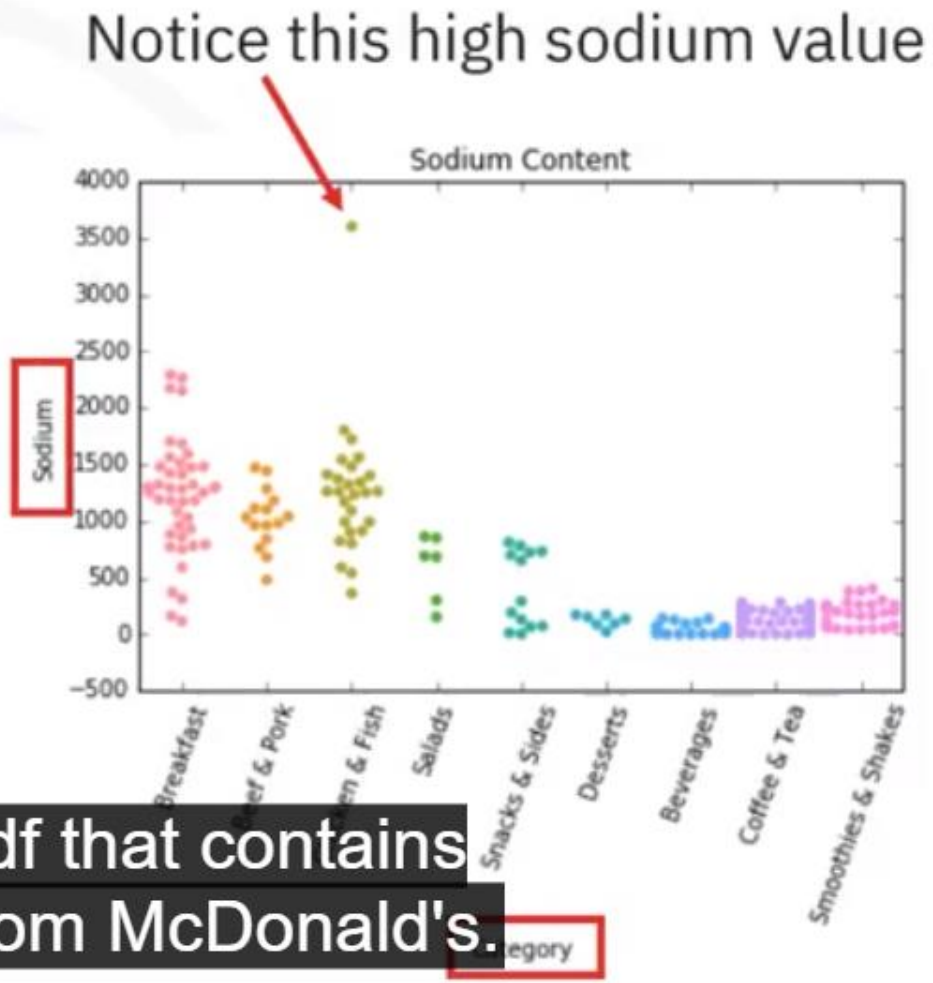
- Main source of sodium is table salt
- Average American eats 5 teaspoons/day
- Sodium mostly added during preparation
- Foods that don’t taste salty may be high in sodium
- Sodium controls fluid balance in our bodies
- Too much sodium may raise blood pressure
- Target less than 2,000 milligrams/day

Which food item has maximum sodium content?

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
### Categorical scatterplots
```

```
plot = sns.swarmplot(x="Category", y='Sodium', data=df)
plt.setp(plot.get_xticklabels(), rotation=70)
plt.title('Sodium Content')
plt.show()
```



will be the data frame, df that contains the nutritional data set from McDonald's.

Which food item has maximum sodium content?

Code 1

```
In [17]: df['Sodium'].describe()

Out[17]: count     260.000000
         mean      495.750000
         std       577.026323
         min         0.000000
         25%       107.500000
         50%       190.000000
         75%       865.000000
         max      3600.000000
         Name: Sodium, dtype: float64
```

To check the values of sodium levels in the food items within the data set,

Which food item has maximum sodium content?

Code 1

```
In [17]: df['Sodium'].describe()

Out[17]: count     260.000000
         mean      495.750000
         std       577.026323
         min         0.000000
         25%       107.500000
         50%       190.000000
         75%       865.000000
         max      3600.000000
         Name: Sodium, dtype: float64
```

Code 2

```
In [24]: df['Sodium'].idxmax()

Out[24]: 82
```

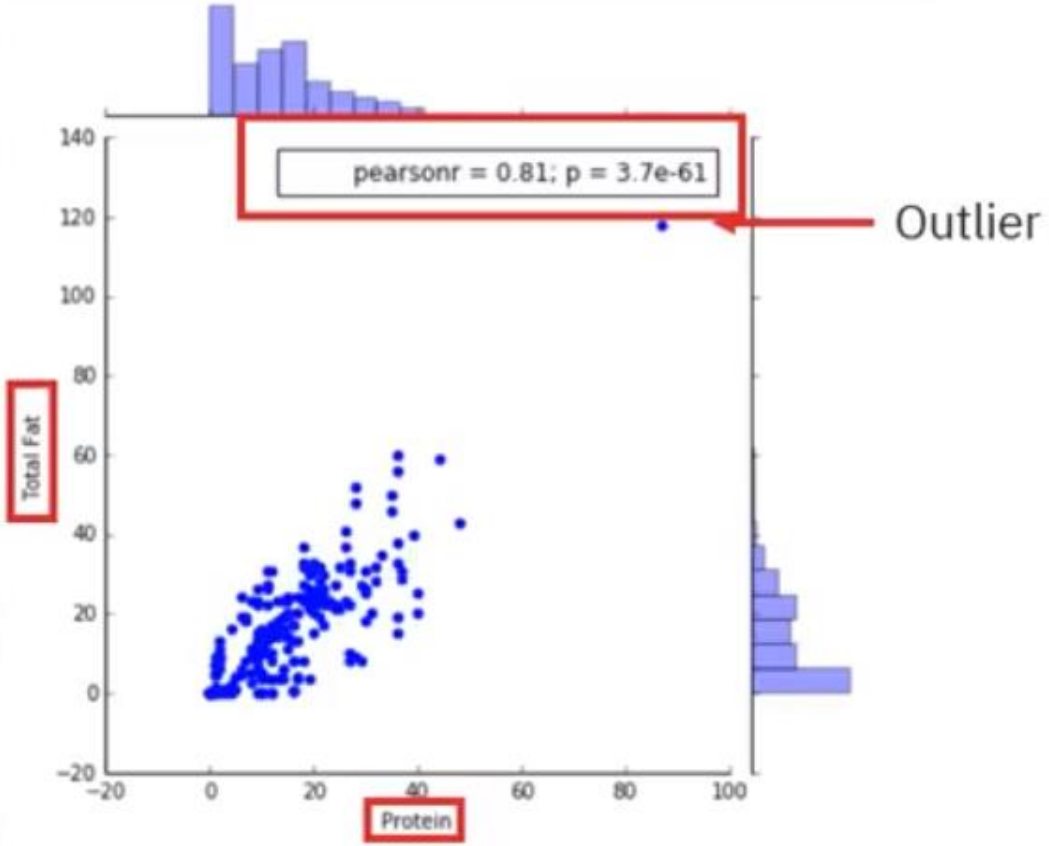
Code 3

```
In [56]: df.at[82,'Item']

Out[56]: 'Chicken McNuggets (40 piece)'
```

a highest sodium content is
Chicken McNuggets, 40 pieces.

Further data exploration using visualizations

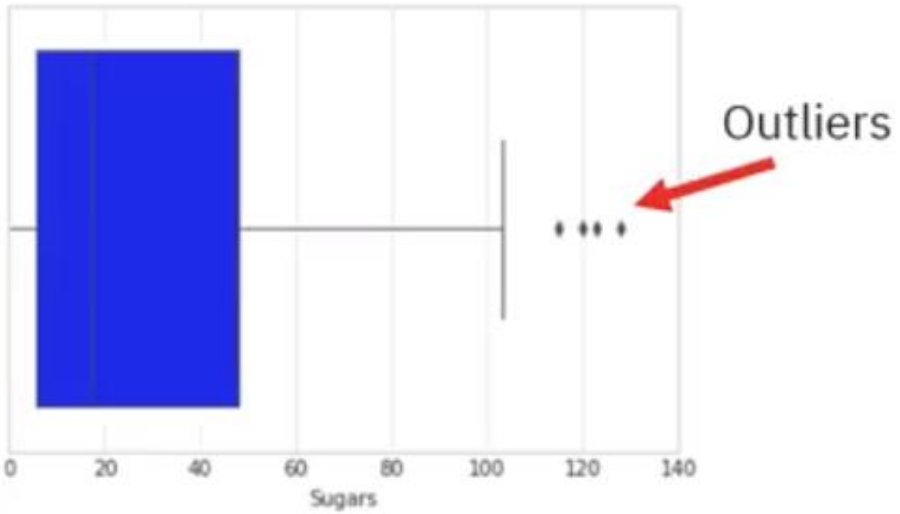
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.jointplot(x="Protein", y='Total Fat', data=df)
plot.show()
```



This is a possible outlier.

Further data exploration using visualizations

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.set_style("whitegrid")
ax = sns.boxplot(x=df["Sugars"])
plot.show()
```



Candies may be among these high sugar content food items on the menu.

Summary

Now you know that:

- SQLite3 is an in-process Python library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.
- The pandas.read_csv function is used to read the database csv file.
- The sqlite3.connect function is used to connect to a database.
- To use pandas to retrieve data from the database tables, load data using the read_sql method and select the SQL Select Query
- A categorical scatterplot is created using the swarmplot() method by the seaborn package.

method by the seaborne package.

Working with CSV files

- Many real data sets are .CSV files
- .CSV: COMMA SEPARATED VALUES

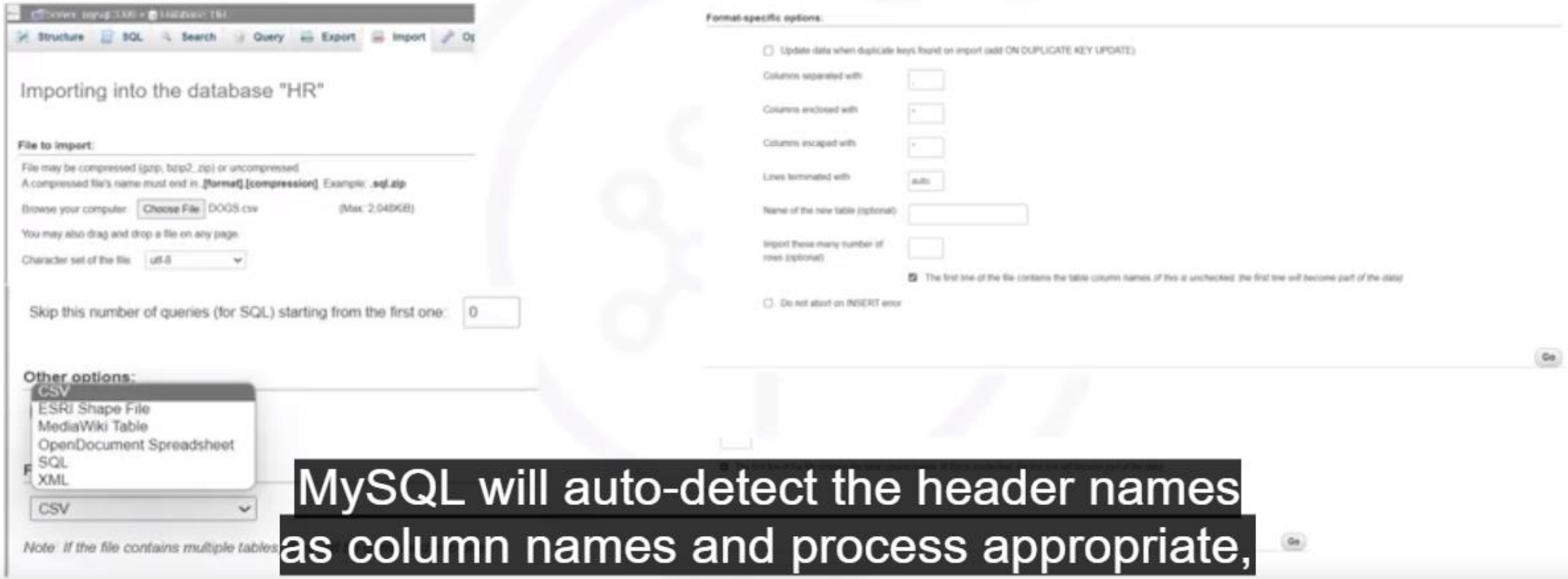
- Example: DOGS.csv

```
Id,Name of Dog,Breed (dominant breed if not pure breed)
1,Wolfie,German Shepherd
2,Fluffy,Pomeranian
3,Huggy,Labrador
```

Although this is a fictional dataset that
contains names of dogs and their breeds,

Column names in CSV row header

MySQL



MySQL will auto-detect the header names as column names and process appropriate,

Querying column names with spaces and special characters

```
Id,Name of Dog,Breed (dominant breed if not pure breed)
1,Wolfie,German Shepherd
2,Fluffy,Pomeranian
3,Huggy,Labrador
```

```
SELECT `Name of Dog`
FROM dogs ;
```

```
SELECT 'Name of Dog'
FROM dogs ;
```

```
SELECT `Breed (dominant
breed if not pure breed)`
FROM dogs ;
```

```
SELECT "Breed (dominant
breed if not pure breed)"
FROM dogs ;
```

will not work with column names.

Splitting queries to multiple lines in Jupyter

Use backslash "\" to split the query into multiple lines:

```
%sql SELECT Id, `Name of Dog`, \
     FROM dogs \
     WHERE `Name of Dog`='Huggy'
```

Or use %%sql in the first row of the cell in the notebook:

```
%%sql
SELECT Id, `Name of Dog`,
     FROM dogs
     WHERE `Name of Dog`='Huggy'
```

This makes the use of backslash at the end of each line redundant.

Using quotes in Jupyter notebooks

```
data = pandas.read_sql(query_statement,
connection_variable)
```

First assign queries to variables:

```
query_statement = ‘SELECT “Name of Dog" FROM dogs'
```

Use a backslash \ as the escape character in cases where the query contains single quotes:

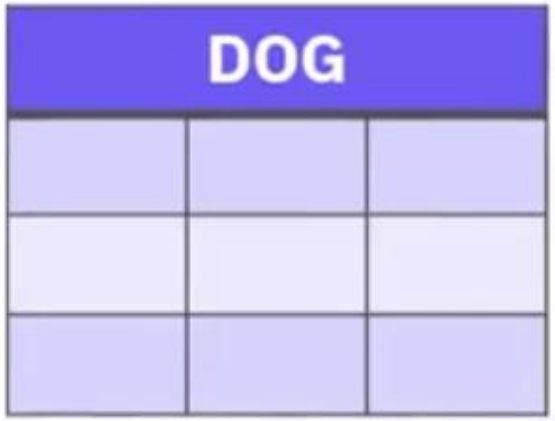
```
query_statement = ‘SELECT * FROM dogs
      WHERE "Name of Dog"=\'Huggy\' '
```

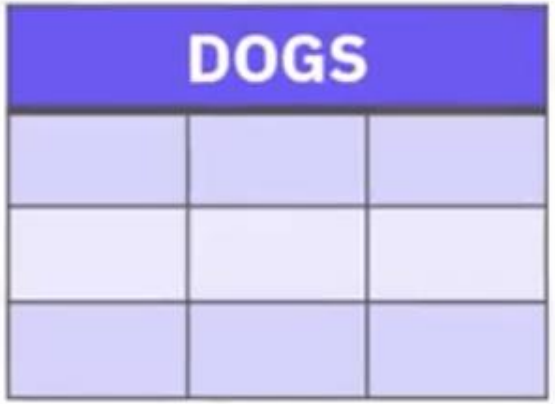
Restricting the # of rows retrieved

To get a sample or look at a small set of rows, limit the result set by using the LIMIT clause:


```
select * from census_data LIMIT 3
```

Getting a list of tables in the database





Getting a list of tables in the database

DB2
SYSCAT.TABLES

SQLite3
sqlite_master

MySQL
SHOW TABLES

Getting a list of tables in the database

Query system catalog to get a list of tables & their properties:

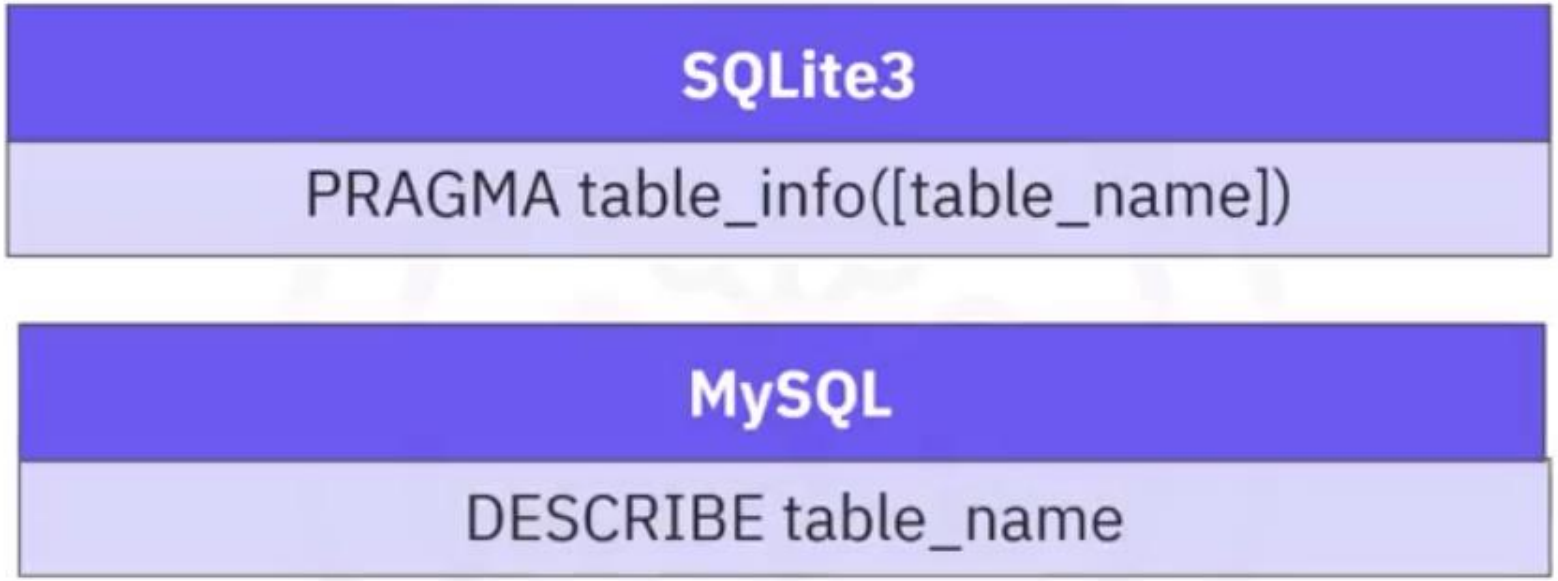
SQLite3

```
SELECT name FROM sqlite_master WHERE type="table"
```

MySQL

```
SHOW TABLES
```

Getting table attributes



Summary

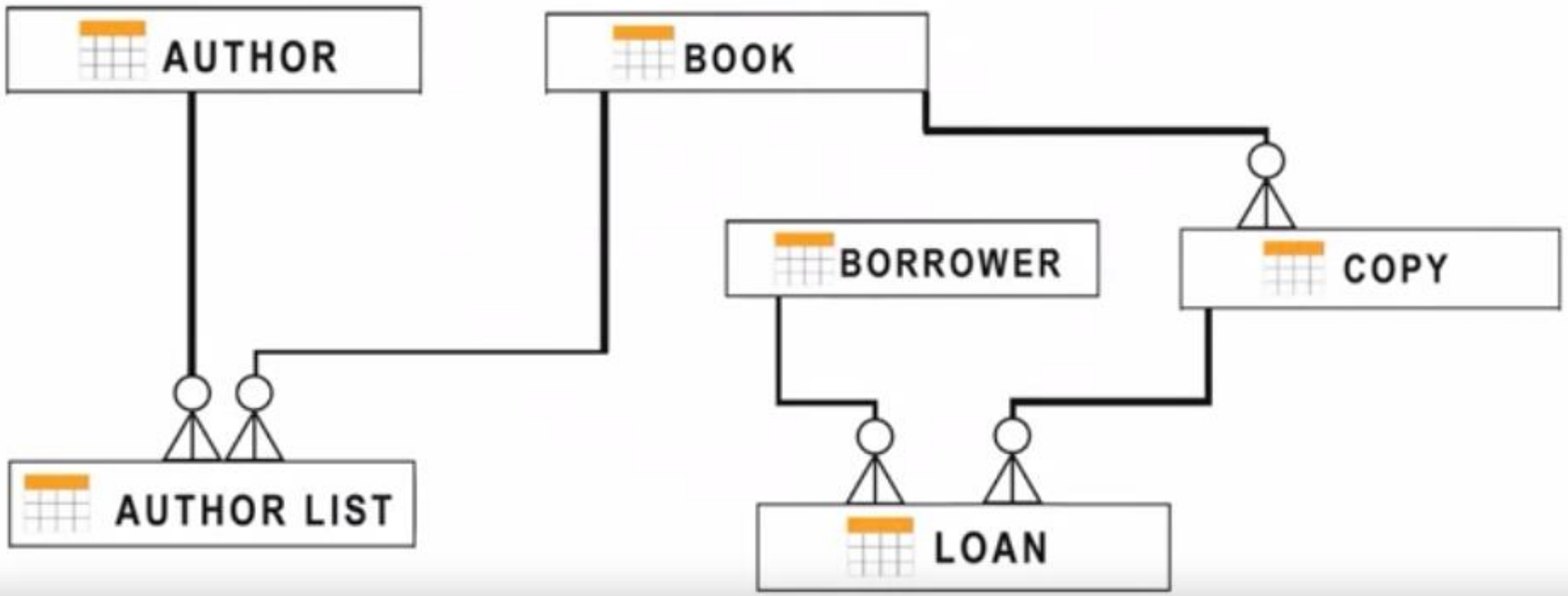
Now you know that:

- Database systems typically contain system or catalog tables from where you can query the list of tables and get their properties.
- To get a list of tables in SQLite3 database, use **SELECT name FROM sqlite_master WHERE type="table"**
- To extract the attributes in SQLite3, use PRAGMA **table_info([table_name])**

Relational model database diagram

JOIN operator:

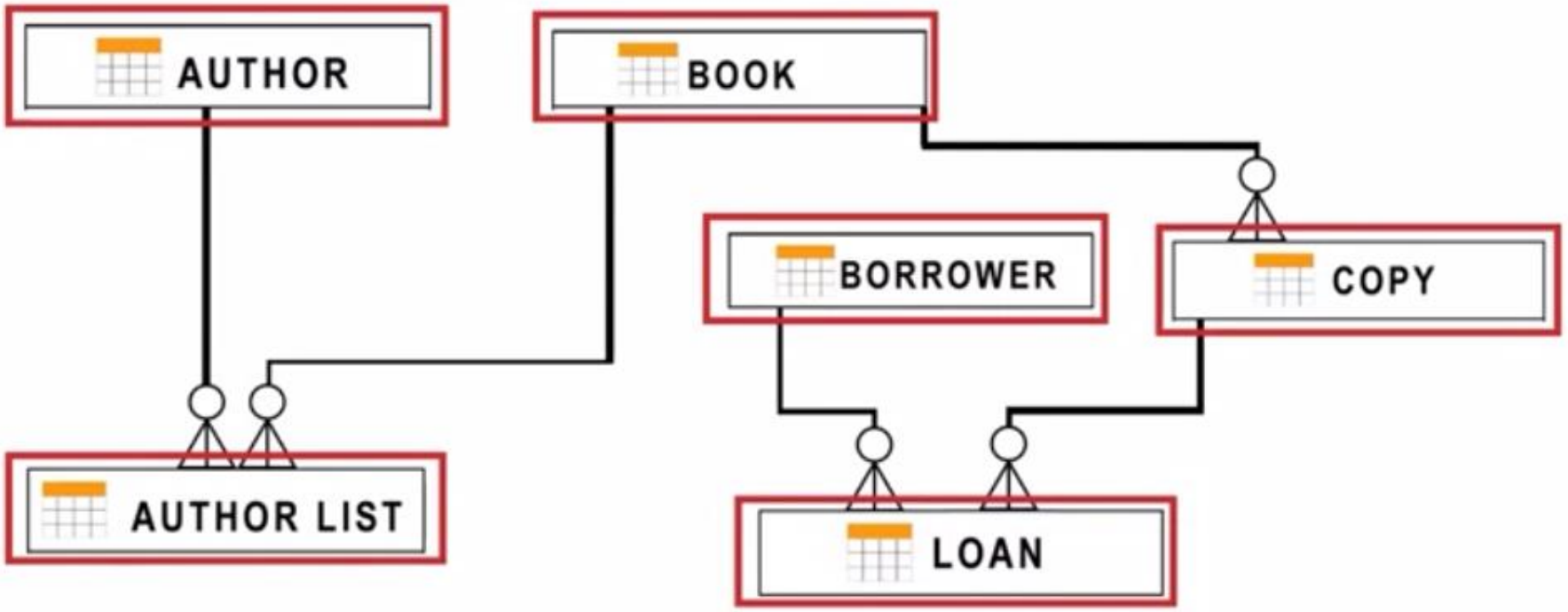
- Combines rows from two or more tables
- Based on a relationship



Relational model database diagram

JOIN operator:

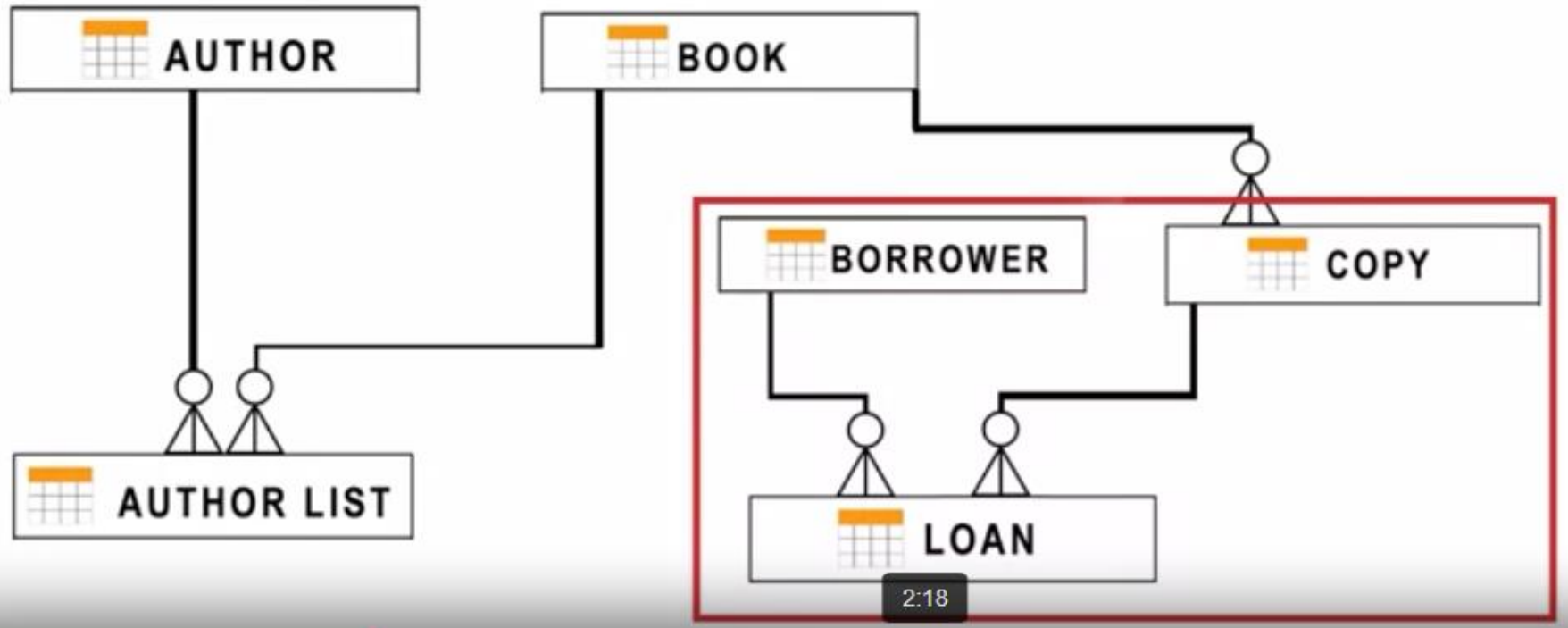
- Combines rows from two or more tables
- Based on a relationship



Relational model database diagram

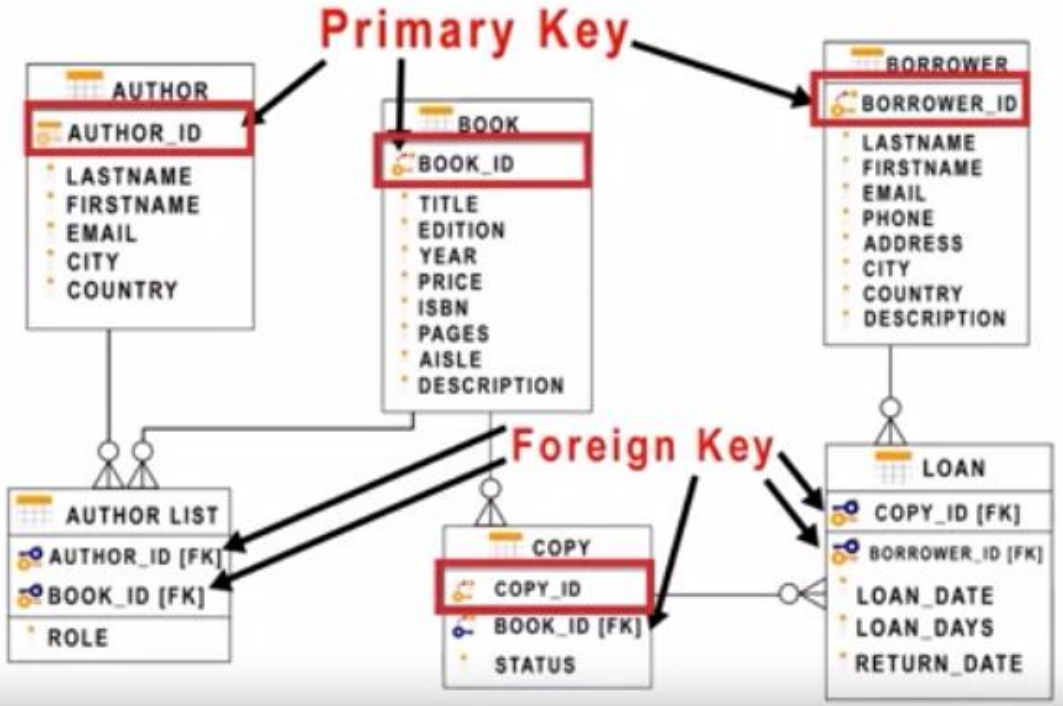
JOIN operator:

- Combines rows from two or more tables
- Based on a relationship



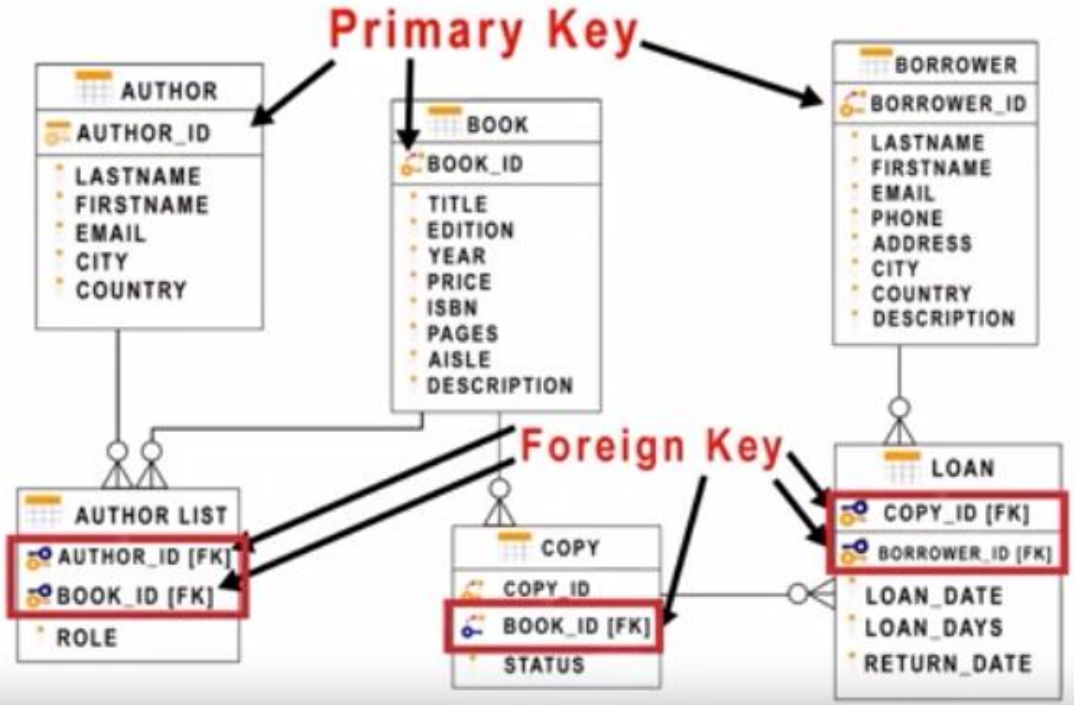
Relational model ER diagram

- Primary key: uniquely identifies each row in a table
- Foreign key: refers to a primary key of another table



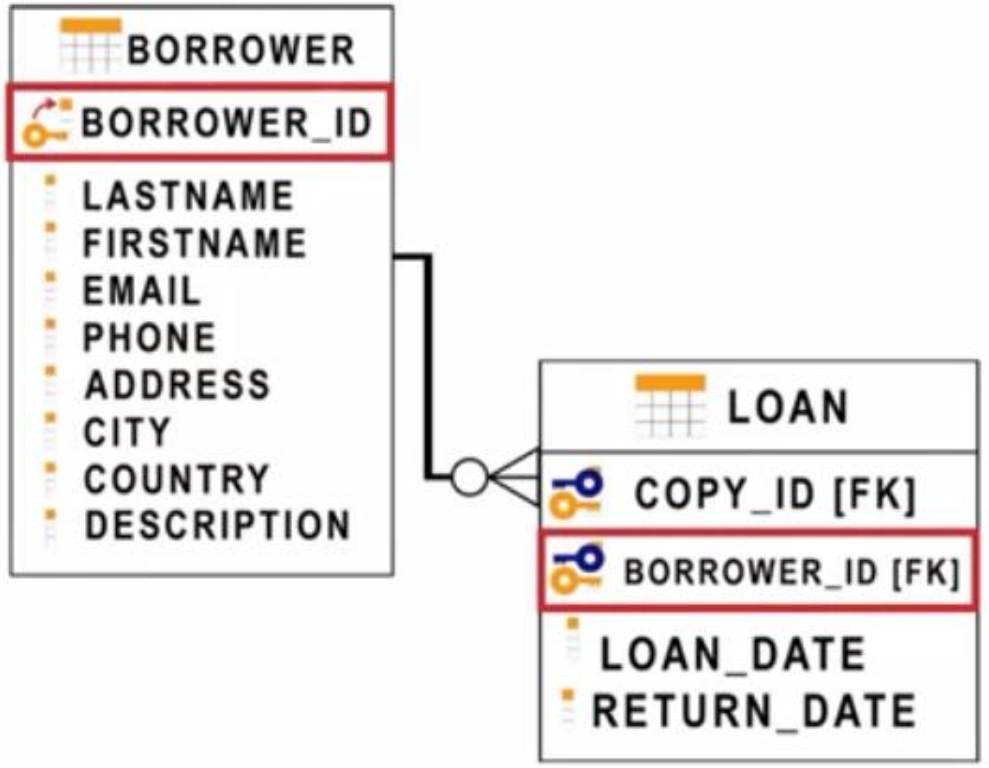
Relational model ER diagram

- Primary key: uniquely identifies each row in a table
- Foreign key: refers to a primary key of another table



Joining tables

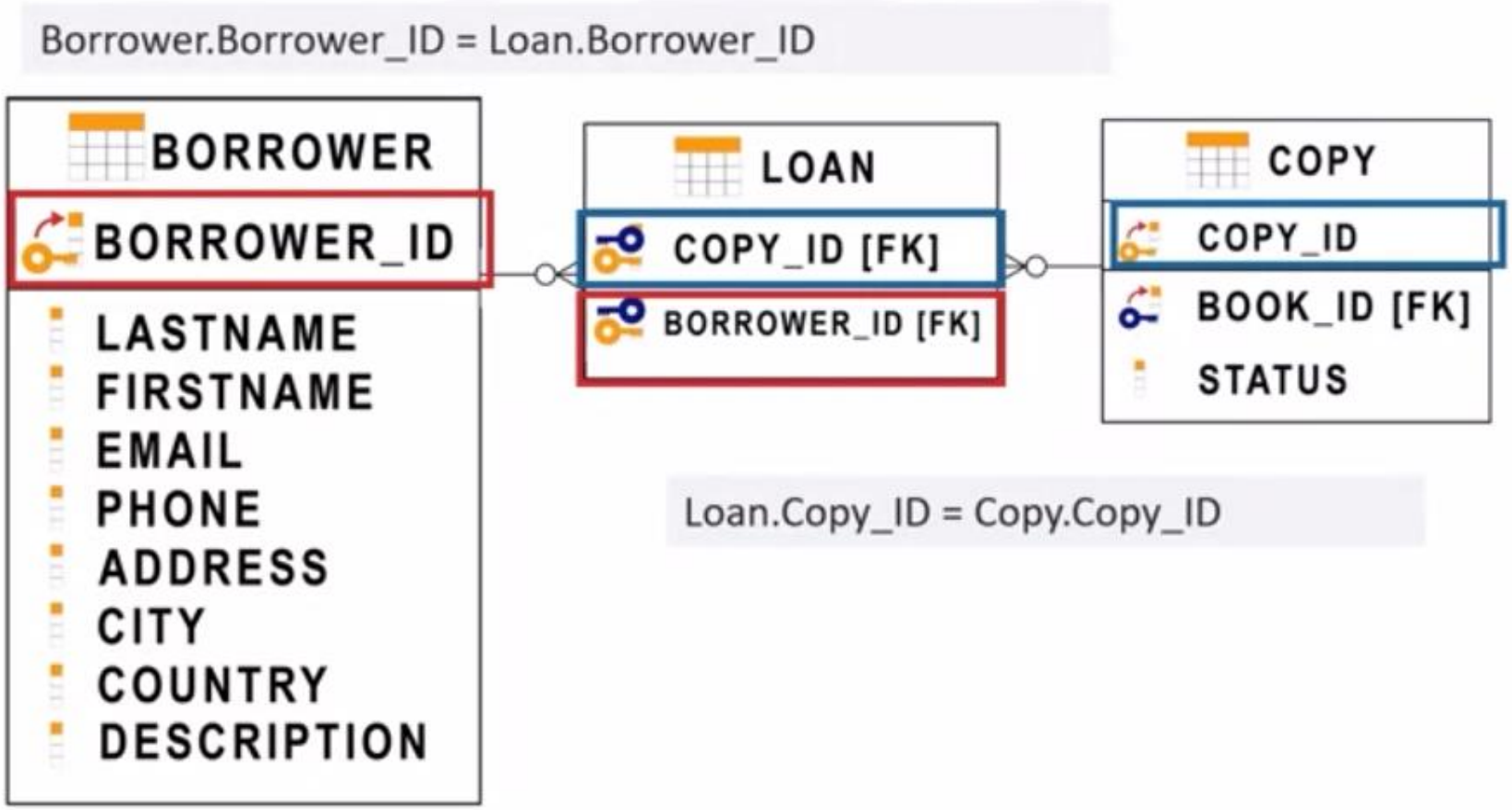
Which borrower has a book out on loan?



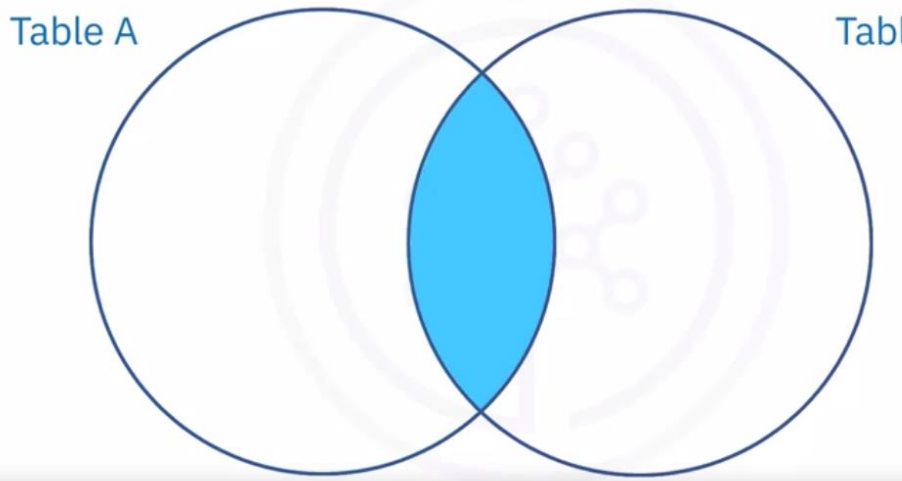
Borrower.Borrower_ID = Loan.Borrower_ID

Joining Three Tables

Which copy of a book does the borrower have on loan?



Types of joins



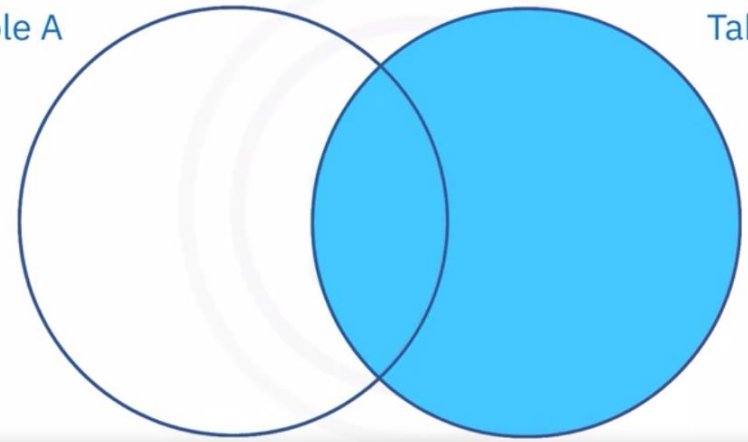
Skills Network

3:04 / 4:06

IBM

Types of joins

To exit full screen, press Esc



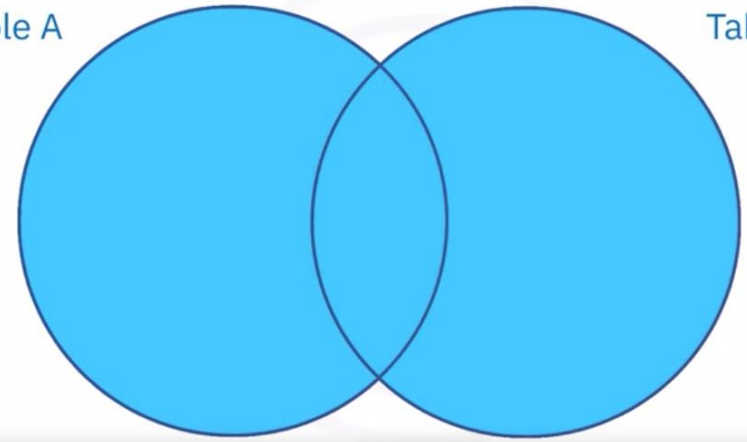
Skills Network

3:09 / 4:06

IBM

Types of joins

To exit full screen, press Esc



Skills Network

3:14 / 4:06

IBM

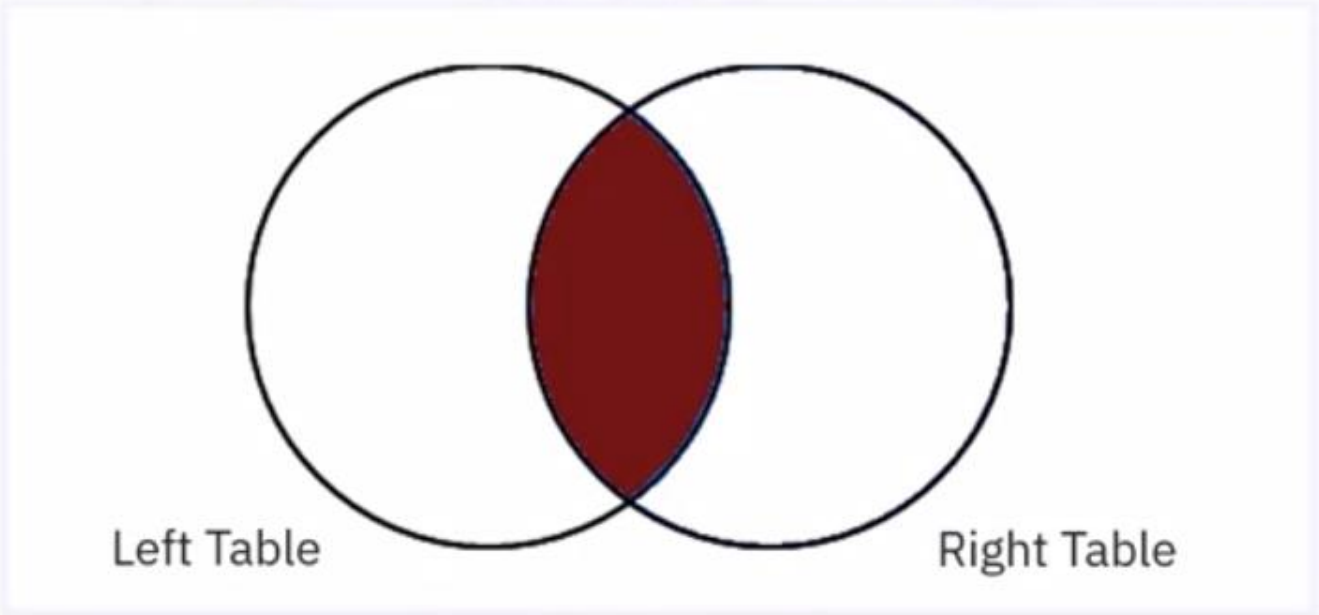
Types of joins

- Inner Join
- Outer Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

There are many varieties of outer joins that you can use to refine your result set.

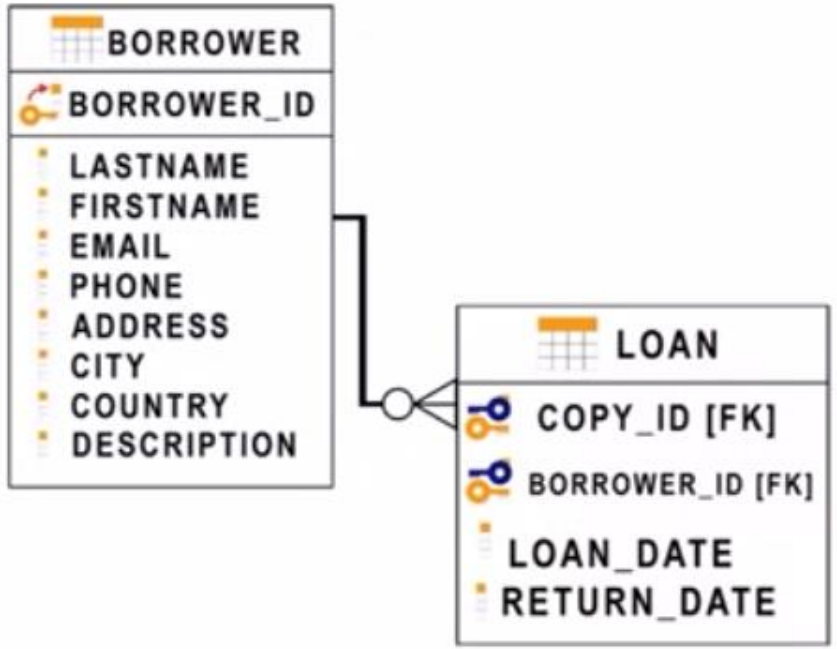
Inner join

- Join operations combine the rows from two or more tables
- Inner join displays matches only



INNER JOIN operator

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```



- In this example, the Borrower table is the Left table
- Each column name is prefixed with an alias to indicate which table each column is associated with

INNER JOIN operator

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

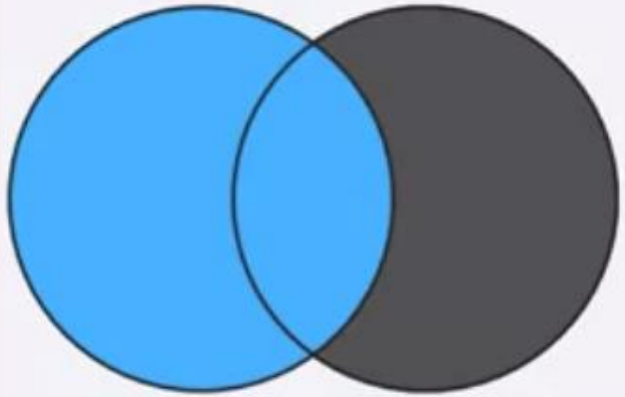
BORROWER_ID	LASTNAME	COUNTRY
D1	SMITH	CA
D2	SANDLER	CA
D3	SOMMERS	CA
D4	ARDEN	CA
D5	XIE	CA
~~D6~~	~~PETERS~~	~~CA~~
~~D7~~	~~LI~~	~~CA~~
~~D8~~	~~WONG~~	~~CA~~
~~D10~~	~~KIEVA~~	~~CA~~

BORROWER_ID	LOAN_DATE
D1	11/24/2010
D2	11/24/2010
D3	11/24/2010
D4	11/24/2010
D5	11/24/2010
~~D9~~	~~11/24/2010~~

Outer joins

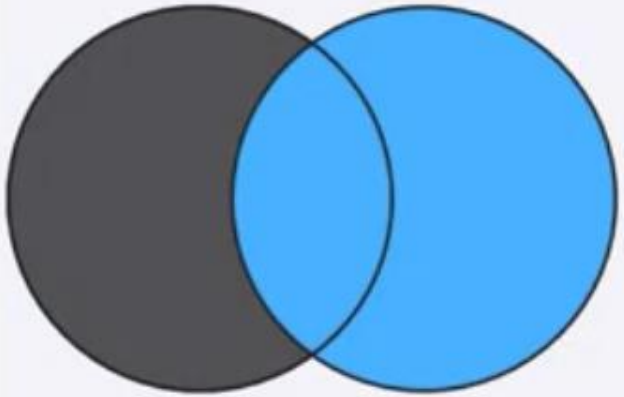
Left Outer Join

All rows from the left table to any matching rows from the right table



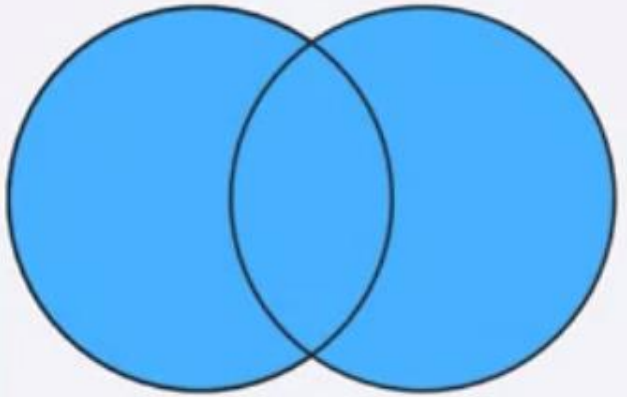
Right Outer Join

All rows from the right table to any matching rows from the left table

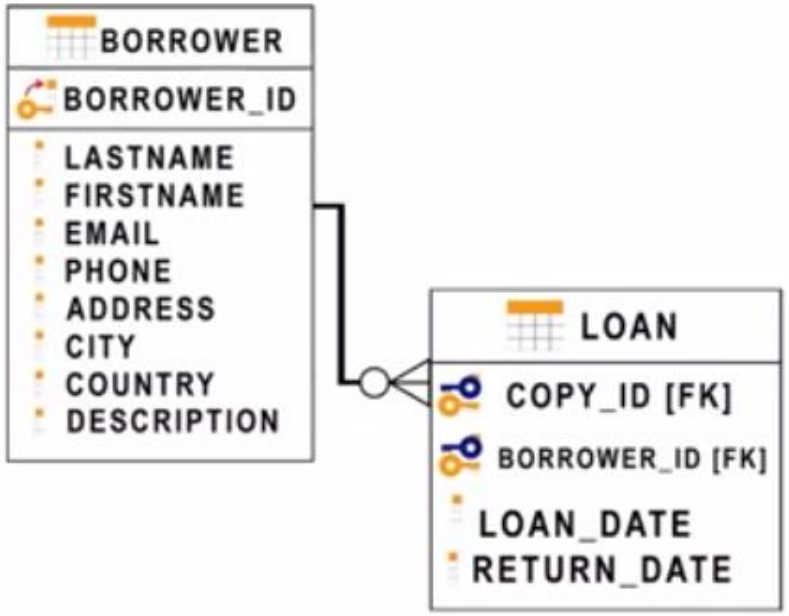


Full Outer Join

All rows from both tables



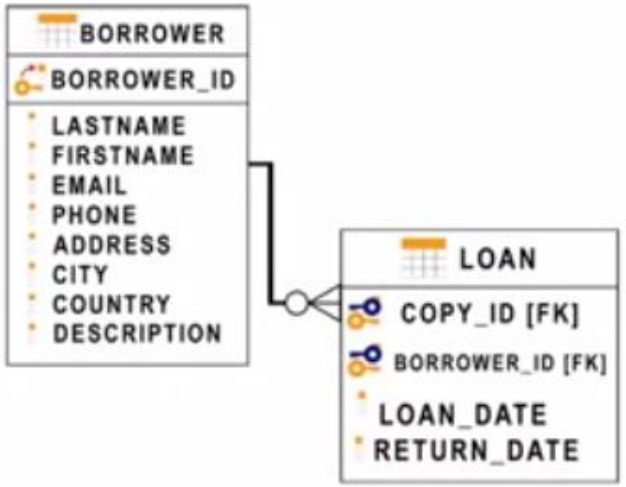
LEFT JOIN operator




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B LEFT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
D6	PETERS	CA	NULL	NULL
D7	LI	CA	NULL	NULL
D8	WONG	CA	NULL	NULL

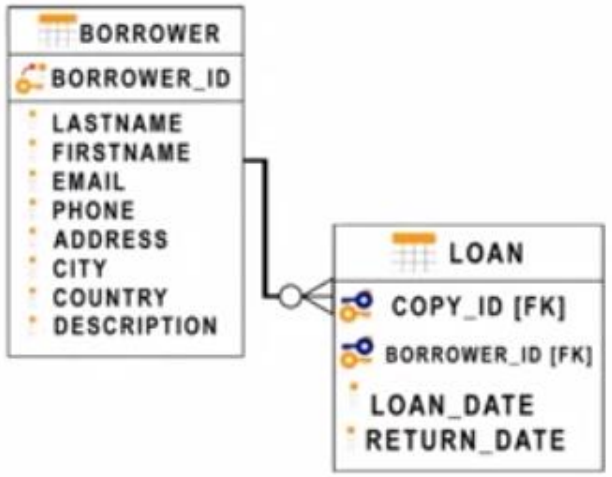
RIGHT JOIN operator




```
SELECT B. BORROWER_ID, B.LASTNAME, B.COUNTRY,
      L. BORROWER_ID, L.LOAN_DATE
FROM BORROWER B RIGHT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
NULL	NULL	NULL	D9	11/24/2010

FULL JOIN operator




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B FULL JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
D6	PETERS	CA	NULL	NULL
D7	LI	CA	NULL	NULL
D8	WONG	CA	NULL	NULL
NULL	NULL	NULL	D9	11/24/2010