Module 1: Basic SQL

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

- 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")

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

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
- Delete

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 <column 1, column 2, ..., column n from Book

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

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

COUNT() - a built-in function that retrieves the number of rows matching the query criteria.

Number of rows in a table:

```
select COUNT(*) from tablename
```

COUNT

Rows in the MEDALS table where Country is Canada:

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

Result:

```
          1
--------------
         29
```

DISTINCT

DISTINCT is used to remove duplicate values from a result set.

Retrieve 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

LIMIT is used for restricting the number of rows retrieved from the database.

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

- After creating a table and inserting data into the table, we can alter the data
  - UPDATE 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 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

```
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

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

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

Deleting Rows from a table

- Remove 1 or more rows from the table:
  - DELETE statement
    - A DML statement used to read and modify data

```
DELETE FROM [TableName]
<WHERE [Condition]>
```

Using the DELETE Statement

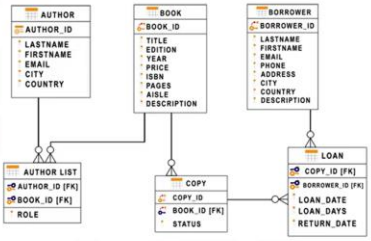
Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

```
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

Relational Model

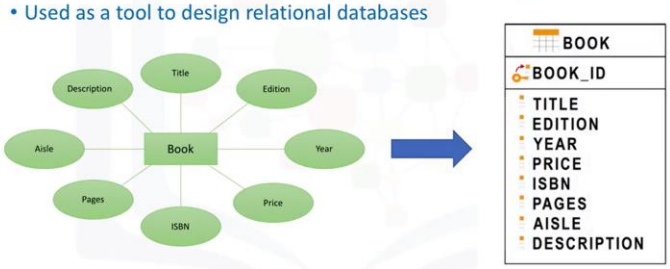
- Most used data model
- Allows for data independence
- Data is stored in a tables



logical data independence - physical data independence - physical storage independence

Entity-Relationship Model

- Used as a tool to design relational databases



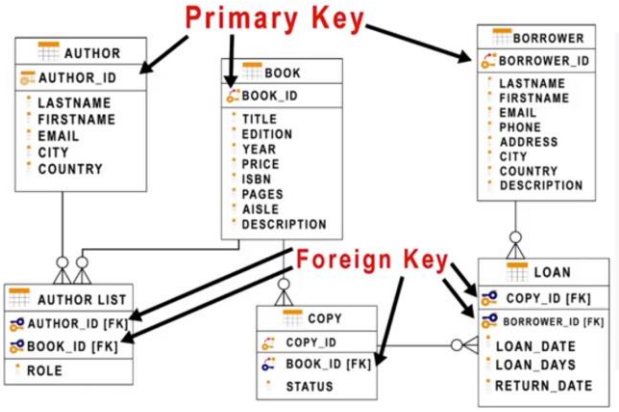
Mapping Entity Diagrams to Tables

- 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



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
   )
```

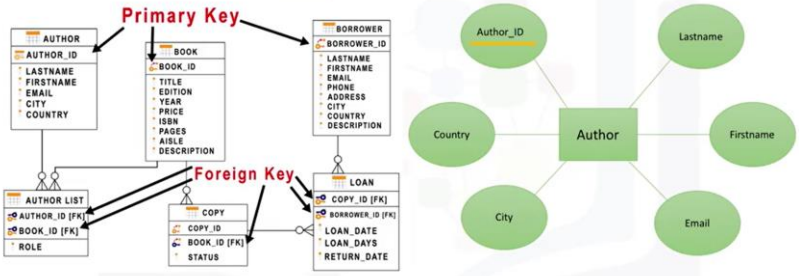
EXAMPLE

- 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 ... ADD COLUMN

```
ALTER TABLE author
  ADD COLUMN telephone_number BIGINT;
```

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

ALTER TABLE ... ALTER COLUMN

```
ALTER TABLE <table_name>
  ALTER COLUMN <column_name> SET DATA TYPE
<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

ALTER TABLE ... DROP COLUMN

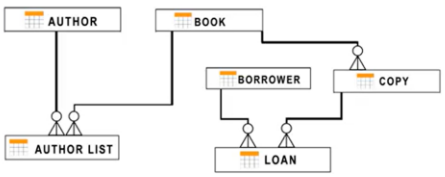
```
ALTER TABLE author
  DROP COLUMN telephone_number;
```

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 <table_name>
    IMMEDIATE;
```

TRUNCATE TABLE

```
TRUNCATE TABLE author
  IMMEDIATE;
```

author_id	lastname	firstname	email	city	country

- The ALTER TABLE statement changes the structure of an existing table, for example to add, modify, or drop columns
- The DROP TABLE statement deletes an existing table
- The TRUNCATE TABLE statement deletes all rows of data in a table

Cloud databases

✓ Ease of Use and Access
- API
- Web Interface
- Cloud or Remote Applications

✓ Scalability & Economics
- Expand/Shrink Storage & Compute Resources
- Pay per use

✓ Disaster Recovery
- Cloud Backups and Geographical Distribution

Examples of Cloud databases

- IBM Db2
- Databases for PostgreSQL
- Oracle Database Cloud Service
- Microsoft Azure SQL Database
- Amazon Relational Database Services (RDS)

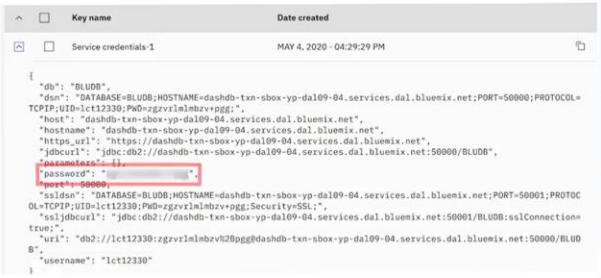
Available as:

- VMs or Managed Service
- Single or Multi-tenant

Database service instances

- DBaaS provides users with access to Database resources in cloud without setting up hardware and installing software.
- Database service instance holds data in data objects / tables
- Once data is loaded, it can be queried using web interfaces and applications

Service credentials



- **Database:** BLUDB
- **Port:** 50000
- **Hostname:** dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net
- **Username:** lct12330
- **Password:** *******

Module 3: Refining Your Results, Functions, Multiple Tables, Subqueries

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.
```

Retrieving rows - using a String Pattern

- WHERE requires a predicate
- A predicate is an expression that evaluates to True, False, or Unknown
- Use the LIKE predicate with string patterns for the search

Example:

- WHERE <columnname> LIKE <string pattern>

```
WHERE firstname LIKE R%
```

Retrieving rows - using a String Pattern

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

FIRSTNAME
---------
Raul
Rav

  2 record(s) selected.
```

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.
```

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
    2 record(s) selected.
```

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

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

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)

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.
```

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)
         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.
```

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)
         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.
```

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

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.

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
```

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
```

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
```

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
```

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
```

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
127.27000000000000000000000000
```

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	5/29/2018
7	Cat	1	44.44	6/11/2018

```
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
```

Date or Time Arithmetic

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

```
Select (RESCUEDATE + 3 DAYS) from PETRESCUE
```

Example 13: Results:

	ID	ANIMAL	QUANTITY	COST	RESCUEDATE	+ 3 DAYS
2018-06-01	1	Cat	9	450.09	5/29/2018	6/1/2018
2018-06-04	2	Dog	3	666.66	6/1/2018	6/4/2018
2018-06-07	3	Dog	1	100	6/4/2018	6/7/2018
2018-06-07	4	Parrot	2	50	6/4/2018	6/7/2018
2018-06-13	5	Dog	1	75.75	6/10/2018	6/13/2018

Date or Time Arithmetic

Special Registers:

```
CURRENT_DATE, CURRENT_TIME
```

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

```
Select (CURRENT_DATE - RESCUEDATE) from PETRESCUE
```

Example 14: Sample result (format YMMDD):

```
10921
```

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	68888.888888888888888888888888
E1003	50000.00	68888.888888888888888888888888
E1004	60000.00	68888.888888888888888888888888
E1005	70000.00	68888.888888888888888888888888
E1006	90000.00	68888.888888888888888888888888
E1007	65000.00	68888.888888888888888888888888
E1008	65000.00	68888.888888888888888888888888
E1009	70000.00	68888.888888888888888888888888
E1010	70000.00	68888.888888888888888888888888

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 ;
```

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

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, etc.)

Tables for Examples in this Lesson

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	Design Team	30003	L0003

Accessing Multiple Tables with Sub-queries

To retrieve only the employee records that correspond to departments in the DEPARTMENTS table:

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

Accessing Multiple Tables with Sub-queries

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
E1007	Mary	Thomas	123412	5/5/1975	F	100 Rose Pl, Gary,IL	650	65000	30003	7
E1008	Bharath	Gupta	123413	5/6/1985	M	145 Berry Ln, Naperville,IL	660	65000	30003	7
E1009	Andrea	Jones	123414	7/9/1990	F	120 Fall Creek, Gary,IL	234	70000	30003	7
E1010	Ann	Jacob	123415	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30004	5

Multiple Tables with Sub-queries

To retrieve only the list of employees from a specific location:

- EMPLOYEES table does not contain location information
- Need to get location info from DEPARTMENTS table

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

Multiple Tables with Sub-queries

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	Jacob	123415	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30004	5

Multiple Tables with Sub-queries

To retrieve the department ID and name for employees who earn more than $70,000:

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

Multiple Tables with Sub-queries

Result:

DEPT_ID_DEP	DEP_NAME
5	Software Group

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

Accessing multiple tables with Implicit Join

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
E1005	Ahmed	Hussain	123410	1/4/1981	M	216 Oak Tree, Geneva,IL	500	70000	30002	2	5	Software Group	30002	L0002
E1006	Nancy	Allen	123411	2/6/1978	F	111 Green Pl, Elgin,IL	600	90000	30002	2	5	Software Group	30002	L0002
E1005	Ahmed			1/4/1981	M			70000			7	Team		
E1006	Nancy	Allen	123411	2/6/1978	F	111 Green Pl, Elgin,IL	600	90000	30003	2	7	Design Team	30003	L0003
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	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30003	5	7	Design Team	30003	L0003

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;
```

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	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30002	5	5	Software Group	30002	L0002

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;
```

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	Design Team
E1010	Software Group

Accessing multiple tables with Implicit Join

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
```

Accessing multiple tables with Implicit Join

Query:

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

Result:

EMP_ID	DEPT_ID_DEP
E1002	5
E1003	5
E1004	5
E1005	2
E1006	2
E1007	7
E1008	7
E1009	7
E1010	5

Module 4: Working with real-world datasets and built-in SQL functions

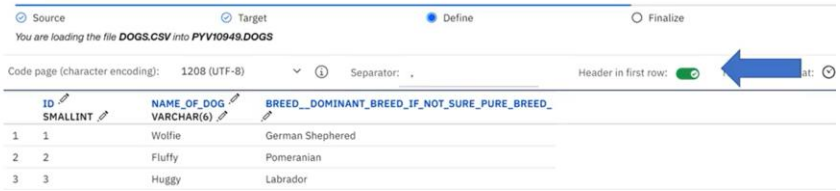
Working with CSV files

- 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
```

Column names in the first row

When the header row in a CSV file contains column names:



Querying column names with mixed (upper and lower) cases

Retrieve ID column from DOGS table. Try:

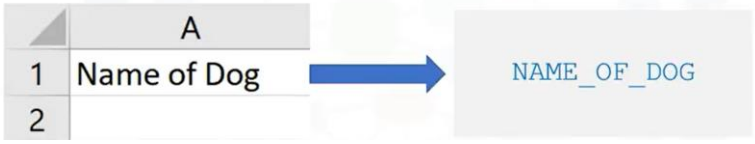
```
select id from DOGS
```

You will get the result as:

ID
1
2

Querying column names with spaces and special characters

By default, spaces are mapped to underscores:



Querying column names with spaces and special characters

Other characters may also get mapped to underscores:

```
select "ID", "NAME_OF_DOG",
"BREED__DOMINANT_BREED_IF_NOT_SURE_PURE_BREED_"
from DOGS;
```

Breed (dominant breed if not sure pure breed)

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

DB2
SYSCAT.TABLES
SQL Server
INFORMATION_SCHEMA.TABLES
Oracle
ALL_TABLES or USER_TABLES
MySQL
SHOW TABLES

Getting a list of tables in the database

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

```
select * from syscat.tables
```


```
select TABSCHEMA, TABNAME, CREATE_TIME
    from syscat.tables
    where TABSCHEMA= 'ABC12345'
```

Getting table properties

```
select TABSCHEMA, TABNAME, CREATE_TIME
     from syscat.tables
     where TABSCHEMA='PYV10949'
```

Getting table properties

```
1 select TABSCHEMA, TABNAME, CREATE_TIME from syscat.tables
2    where tabschema='PYV10949';
3
4
```

Result set 1 — Find

TABSCHEMA	TABNAME	CREATE_TIME
PYV10949	SPACEX	2021-11-16 10:41:55.677651
PYV10949	DOGS	2021-11-25 09:37:23.826177
PYV10949	CENSUS	2021-11-24 06:17:40.214869
PYV10949	CRIME	2021-11-24 06:20:15.086512
PYV10949	SCHOOLS	2021-11-24 06:21:16.987031

Getting a list of columns in the database

To obtain the column names query syscat.columns:

```
select * from syscat.columns
       where tabname = 'DOGS'
```

To obtain specific column properties:

```
select distinct(name), coltype, length
       from sysibm.syscolumns
       where tbname = 'DOGS'
```

Column info for a real table

```
1
2 select distinct(name), coltype, length
3       from sysibm.syscolumns
4       where tbname = 'CHICAGO_CRIME_DATA';
```

NAME	COLTYPE	LENGTH
ARREST	VARCHAR	5
BEAT	SMALLINT	2
BLOCK	VARCHAR	35
CASE_NUMBER	VARCHAR	8
COMMUNITY_AREA_NUMBER	SMALLINT	2
DATE	VARCHAR	256
DESCRIPTION	VARCHAR	46
DISTRICT	SMALLINT	2

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

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>;
```

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	LASTNAME	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

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

Benefits of stored procedures

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

CREATE PROCEDURE statement

```
CREATE PROCEDURE UPDATE_SAL (IN empNum CHAR(6), IN rating SMALLINT)
  LANGUAGE SQL
  BEGIN
    IF rating = 1 THEN
      UPDATE employees
        SET salary = salary * 1.10
        WHERE emp_id = empNum;
    ELSE
      UPDATE employee
        SET salary = salary * 1.05
        WHERE emp_id = empNum;
    END IF;
  END
```

Working with stored procedures

Call from:

- External applications
- Dynamic SQL statements

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

What is a transaction?

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

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	100
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	100
B002	James	13450
B003	Shoe Shop	124200
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';
```

Transaction example

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	100
B002	James	13450
B003	Shoe Shop	124200
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	11	200
High heels	8	600
Brogues	10	150
Trainers	14	300

If any of these UPDATE statements fail, the whole transaction must fail

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

```
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’;
```

ACID commands

- BEGIN
  - Start the ACID transaction
- COMMIT
  - All statements complete successfully
  - Save the new database state

```
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’;
COMMIT
```

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

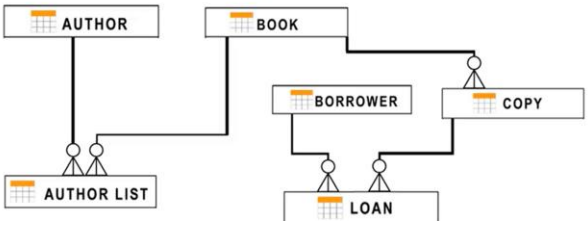
```
void main ()
 {
   EXEC SQL  UPDATE BankAccounts
         SET Balance = Balance – 200
         WHERE AccountName = 'Rose';
   EXEC SQL  UPDATE BankAccounts
         SET Balance = Balance + 200
         WHERE AccountName = 'Shoe Shop';
  EXEC  SQL  UPDATE  ShoeShop
         SET Stock = Stock - 1
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

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

Relational model database diagram

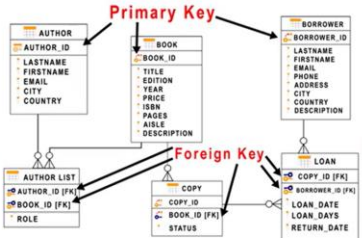
JOIN operator:

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



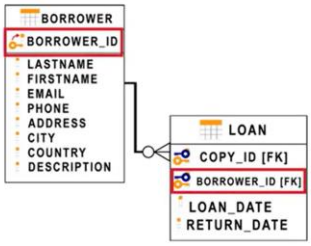
Relational model ER diagram

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



Joining tables

Which borrower has a book out on loan?

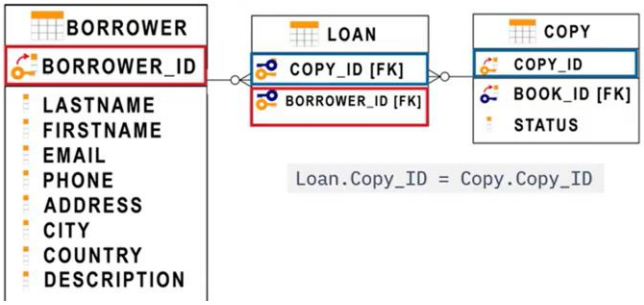



```
Borrower.Borrower_ID = Loan.Borrower_ID
```

Joining Three Tables

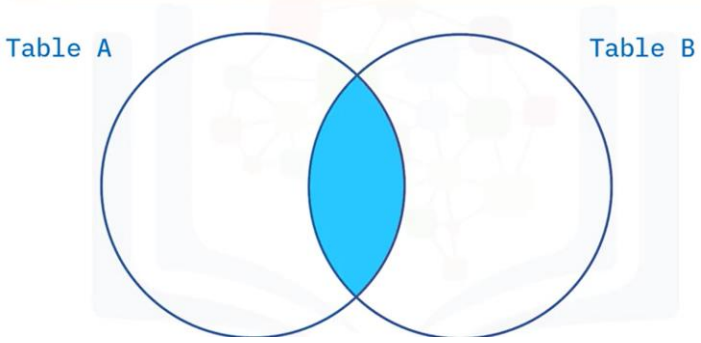
Which copy of a book does the borrower have on loan?

```
Borrower.Borrower_ID = Loan.Borrower_ID
```

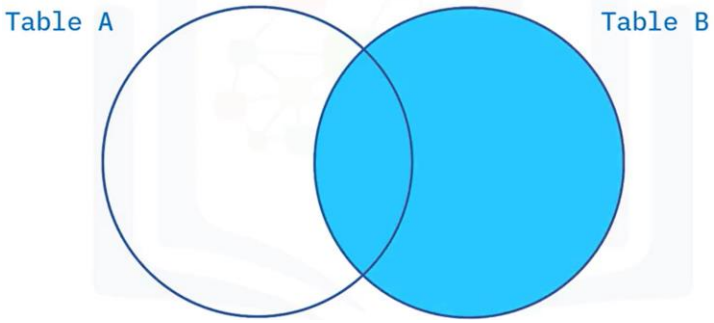



```
Loan.Copy_ID = Copy.Copy_ID
```

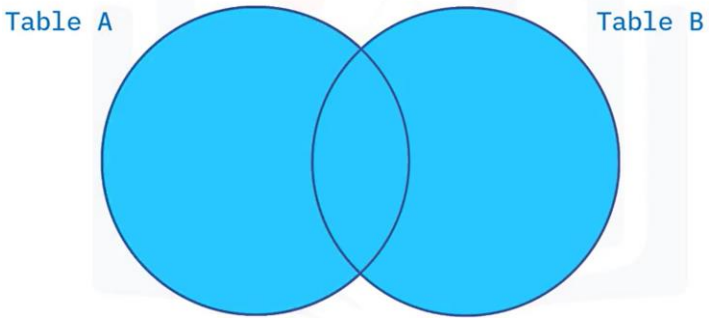
Types of joins



Types of joins



Types of joins

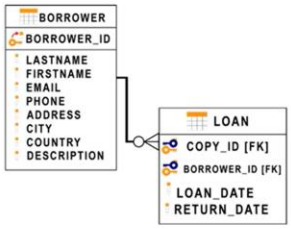


Types of joins

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

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~~

INNER JOIN operator

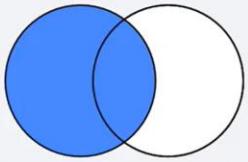
```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

BORROWER_ID	LASTNAME	COUNTRY	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

Outer joins

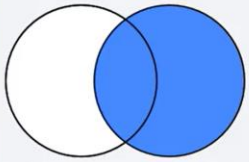
Left Outer Join

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



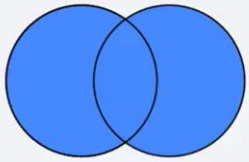
Right Outer Join

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

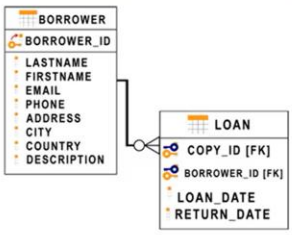


Full Outer Join

All rows from both tables



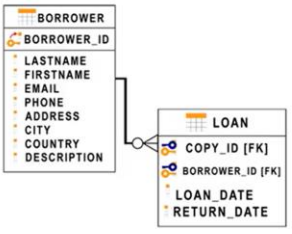
LEFT JOIN Operator




```
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
```

In this example, the Borrower table is the Left table

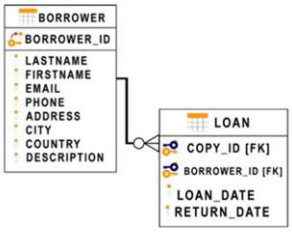
LEFT JOIN Operator




```
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

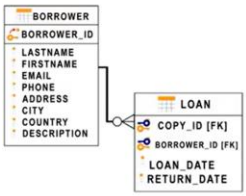
RIGHT JOIN Operator




```
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
```

In this example, the Loan table is the right table

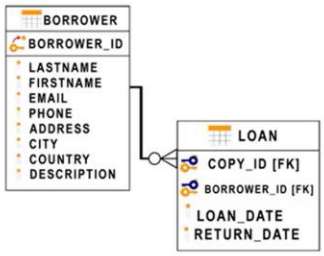
RIGHT JOIN Operator




```
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

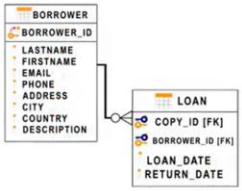
FULL JOIN Operator




```
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
```

In this example, the Borrower table is the Left table

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