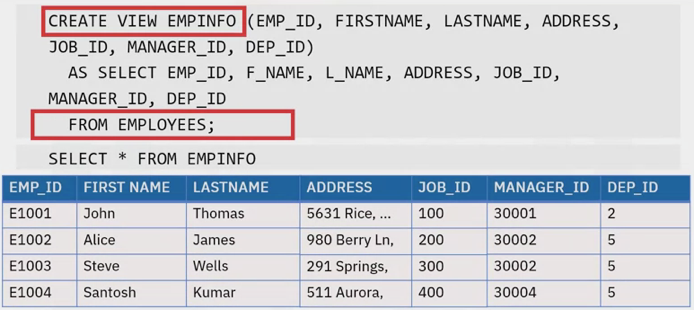
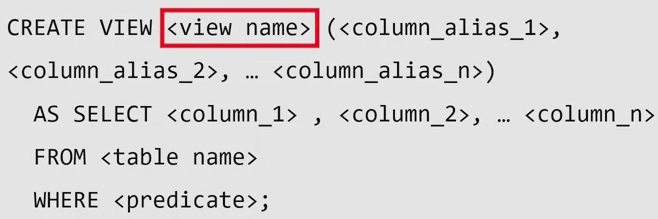
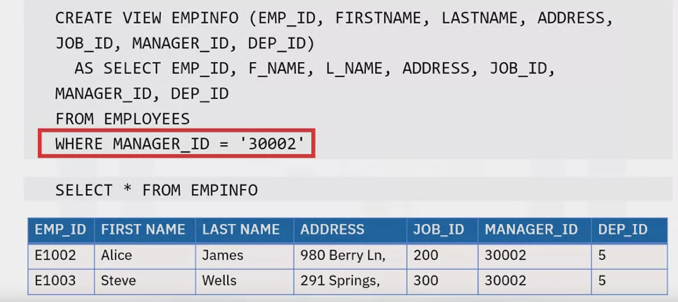
**DATABASES AND SQL FOR DATA SCIENCE - WEEK 6 HONORS MODULE DATA ENGINEERING**

**Views, Stored Procedures, and Transactions**

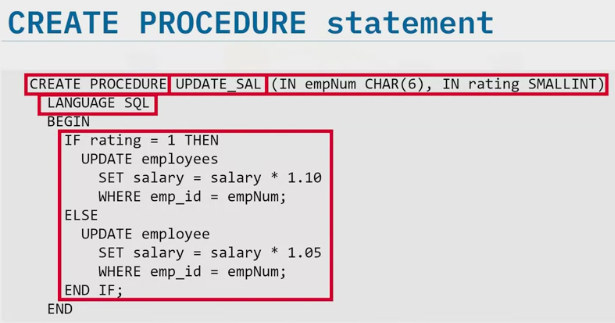
Views

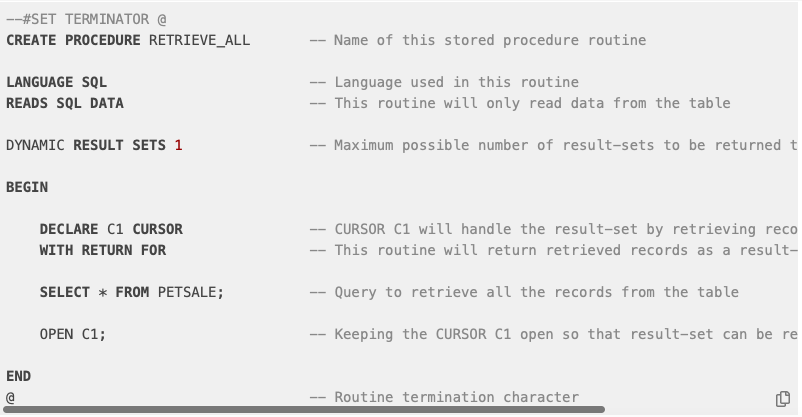
1. Alternative way of representing data that exists in one or more table or views
2. Can include specified columns from multiple base tables and existing views
3. Once created, can be queried like a table
4. Only the definition of the view is stored, not the data
5. Views can:
   1. Show a selection of data for a given table
   2. Combine two or more tables
   3. Simplify access to data
   4. Show only portions of data in the table
   5. Display non-sensitive data
   6. Drop View to drop a view

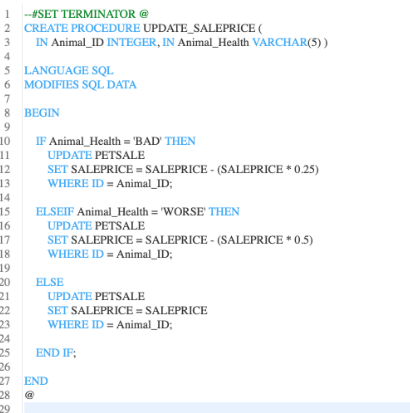




Stored Procedures

1. A set of SQL statements stored and executed on the database server
   1. Stored procedures can be written in different languages
   2. Accept information in form of parameters (CRUD)
   3. Return results to the client
2. Benefits:
   1. Reduction in network traffic – only one call is needed
   2. Improvement in performance
   3. Reuse of code – multiple apps can use the same stored procedure
   4. Increase in security
      1. Do not need to expose all of the table to client side developers
      2. Use server side logic to validate data before accepting it into the system
3. Create procedure, specifying name of procedure and any parameters it will take (e.g. take empNum and increase salary depending on the rating)
4. Language you are using
5. Procedural logic enclosed in begin and end statement
6. Call stored procedures from
   1. External applications
   2. Dynamic SQL statements (e.g. “call update\_sal (‘E1001’, 1))





--#SET TERMINATOR @

CREATE PROCEDURE UPDATE\_SALEPRICE (

IN Animal\_ID INTEGER, IN Animal\_Health VARCHAR(5) ) -- ( { IN/OUT type } { parameter-name } { data-type }, ... )

LANGUAGE SQL -- Language used in this routine

MODIFIES SQL DATA -- This routine will only write/modify data in the table

BEGIN

IF Animal\_Health = 'BAD' THEN -- Start of conditional statement

UPDATE PETSALE

SET SALEPRICE = SALEPRICE - (SALEPRICE \* 0.25)

WHERE ID = Animal\_ID;

ELSEIF Animal\_Health = 'WORSE' THEN

UPDATE PETSALE

SET SALEPRICE = SALEPRICE - (SALEPRICE \* 0.5)

WHERE ID = Animal\_ID;

ELSE

UPDATE PETSALE

SET SALEPRICE = SALEPRICE

WHERE ID = Animal\_ID;

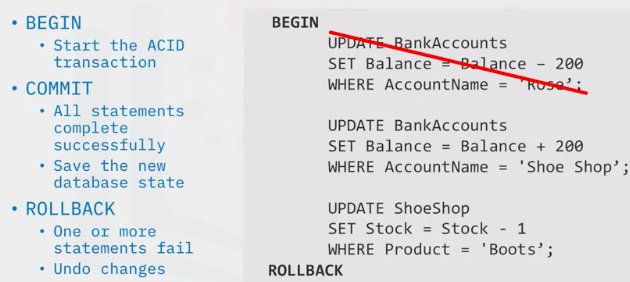
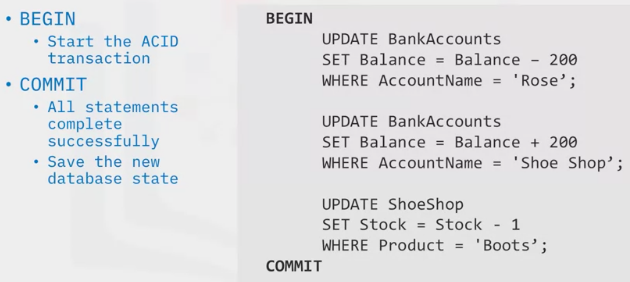
END IF; -- End of conditional statement

END

@ -- Routine termination character

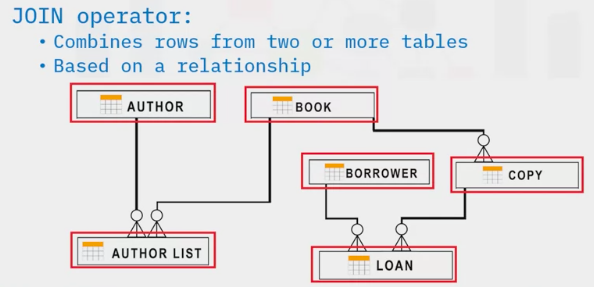
ACID Transactions

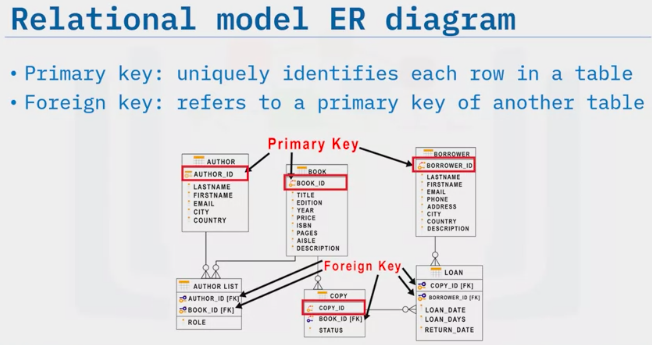
1. Indivisible unit of work
2. Consists of one or more SQL statements
3. Either all happens or none
4. ACID:
   1. Atomic: All changes must be performed successfully or not at all
   2. Consistent: Data must be in a consistent state before and after transaction
   3. Isolated: No other process can change the data while the transaction is running
   4. Durable: The changes made by the transaction must persist
      1. Commit: Successful / / Rollback: Unsuccessful

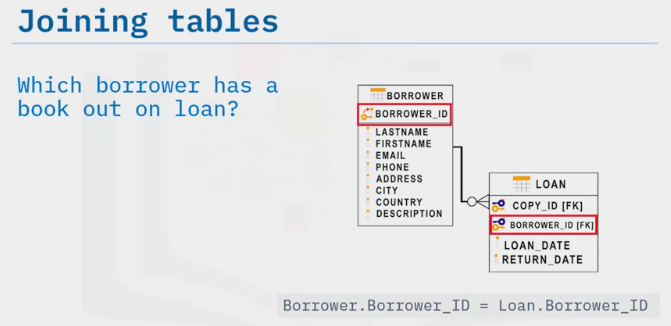
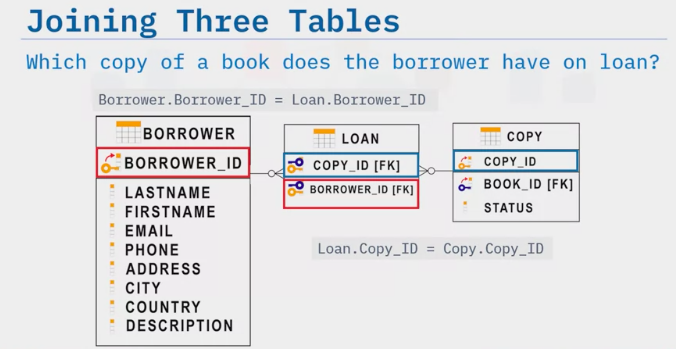


1. Can be issued by other languages:
   1. Java, C, R and Python
   2. Requires use of database specific API’s or connectors
   3. Use the EXEC SQL command to execute SQL statements from code
2. Exec SQL command:
   1. “Begin” is implicit – does not need t be called out explicitly
   2. Incorporating EXEC SQL commands into your code gives you the opportunity to create error checking routines that in turn controls whether the transaction is committed or rolled back

Join Overview

1. If you wanted to know which borrower had which book on lone, you will need data from three tables (borrower, copy and loan). Join operator needed.
2. Identify the relationship (i.e. column or columns to use as a link between the tables)

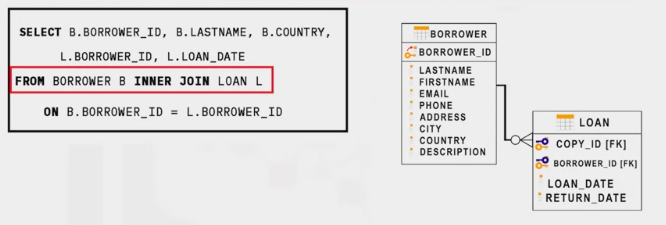


1. Join the info from the borrower table and Loan table by matching borrower ID
2. Then join the information from the loan table and copy table by matching the copy\_ID’s
3. Types of Joins
   1. Inner Joins: Displays only the rows from two tables that have matching value in a common column. Usually the primary key that exists as a foreign key in the second table
   2. Outer Join: Return matching rows and row from one or the other table that don’t match
      1. Left outer join
      2. Right outer join
      3. Full outer join

Inner Join

1. Displays only the matches in a table
   1. Usually the primary key of one table that exists as a foreign key in the other table

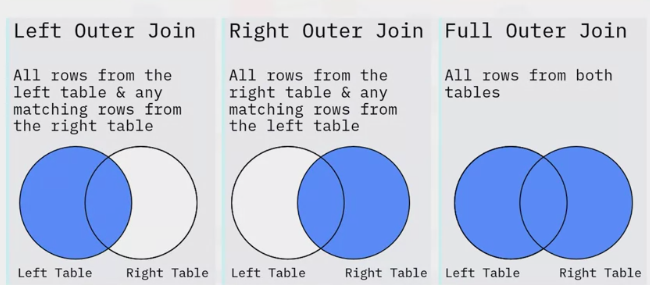


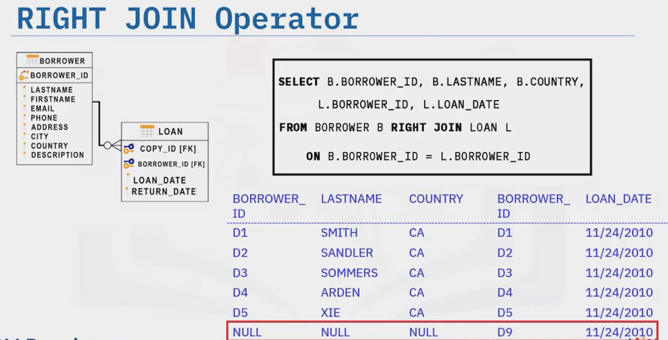
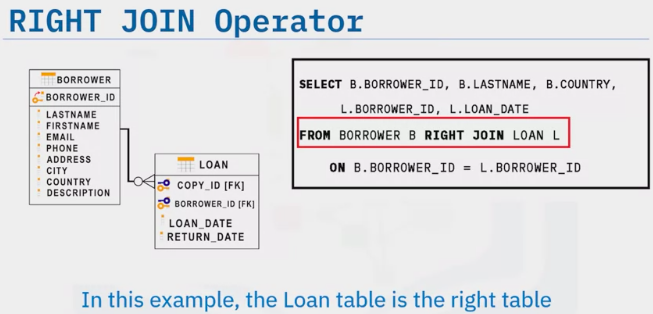
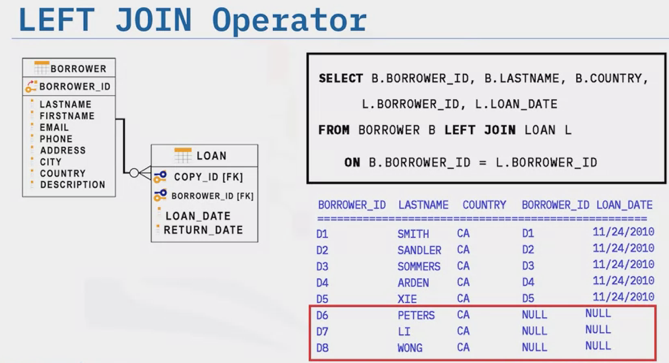
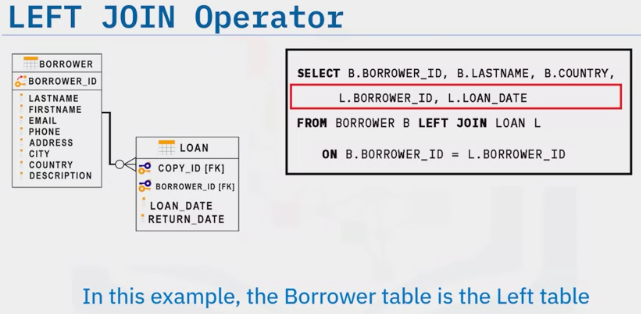
1. “From clause” you specify the join
2. Each column name is prefixed with either “B” or “L”. This is known as an alias

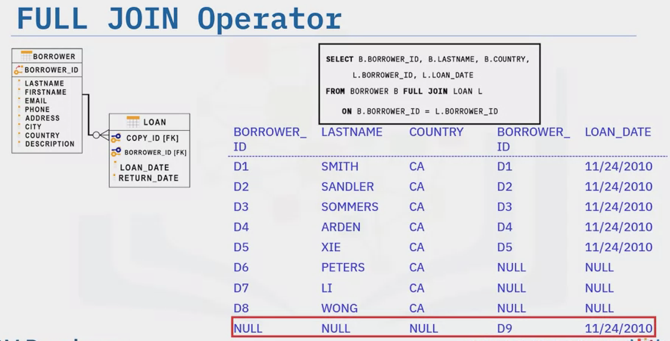
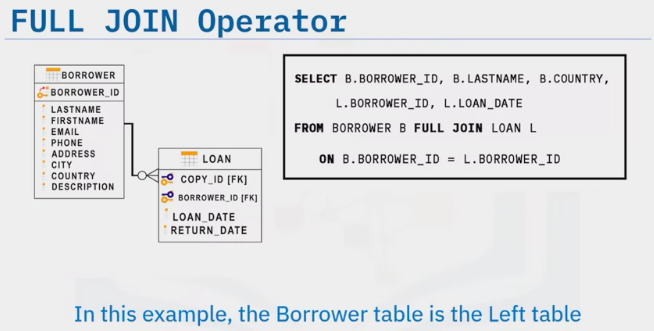


Outer Joins

1. Outer Joins
   1. Return the rows from each table that have matching values in the join columns
   2. Return the rows that do not have a match between the tables
   3. SQL offers 3 types of outer join:







Cross Join (Cartesian Join)

SELECT column\_name(s)

FROM table1

CROSS JOIN table2;

Inner Join

SELECT column\_name(s)

FROM table1

INNER JOIN table2

ON table1.column\_name = table2.column\_name;

WHERE condition;

Left Outer Join

SELECT column\_name(s)

FROM table1

LEFT OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;

Right Outer Join

SELECT column\_name(s)

FROM table1

RIGHT OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;

Full Outer Join

SELECT column\_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;

Self Join

SELECT column\_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;