USING DATABASES FOR DATA ANALYTICS

WHAT IS A DATABASE?

database

A database is a collection of data in a structured format. In principle, databases can be stored on paper or even clay tablets. In practice, however, modern databases are invariably stored on computers.

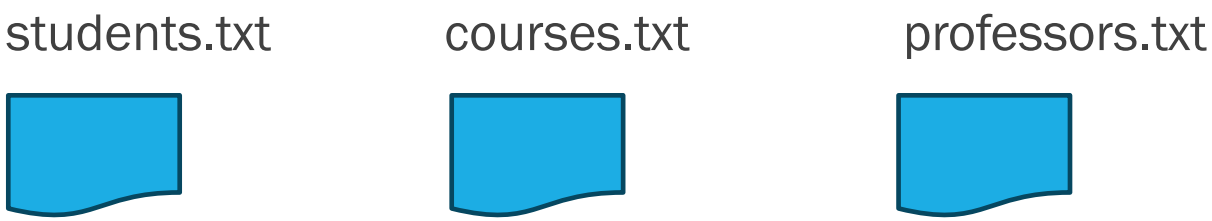
database system / database management system / DBMS

A database system, also known as a database management system or DBMS, is software that reads and writes data in a database. Database systems ensure data is secure, internally consistent, and available at all times. These functions are challenging for large databases with many users, so database systems are complex.

BYU

CAN WE ACCESS DATA WITHOUT A DATABASE MANAGEMENT SYSTEM?

Sure, we can! Start by storing the data in flat files on your PC:



Now write programs to implement specific tasks

BYU

DOING IT WITHOUT A DBMS…

Write a program to do the following:

- Enroll “Mary Johnson” in “CS 452”:

1. Read ‘students.txt’
2. Read ‘courses.txt’
3. Find & update the record “Mary Johnson”
4. Find & update the record “CS 452”
5. Write “students.txt”
6. Write “courses.txt”

BYU

DRAWBACKS OF USING FILE SYSTEMS TO STORE DATA:

- Data redundancy and inconsistency
  - Multiple file formats, duplication of information in different files
- Difficulty accessing data
  - Need to write a new program to carry out each new task
- Concurrent Access by multiple users (How do you assure consistency?)
- Integrity problems (What if there is a failure in the middle of an update?)
  - Integrity constraints (e.g. account balance > 0) become part of program code
  - Hard to add new constraints or change existing ones

BYU

FUNCTIONALITY OF A DBMS

The programmer sees :

- Data Definition Language – DDL
  - Create tables
- Data Manipulation Language - DML
  - query language

Behind the scenes the DBMS has:

- Query optimizer
- Query engine
- Storage management
- Transaction Management (concurrency, recovery)

BYU

DATA DEFINITION LANGUAGE (DDL)

- Specification notation for defining the database schema:

```
create table account (
account-number  char(10),
balance  integer);
```

- DDL compiler generates a set of tables stored in a *data dictionary*
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
    - Information about tables, columns, data types, and constraints.
  - Data *storage and definition* language
    - Specifies how data is stored and accessed.
    - Includes indexing, partitioning, and other performance-related features.

BYU

DATA MANIPULATION LANGUAGE (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as **query language**
- Two classes of languages
  - Procedural – user specifies what data is required and how to get those data
  - **Nonprocedural – user specifies what data is required without specifying how to get those data**
- **SQL is the most widely used query language**

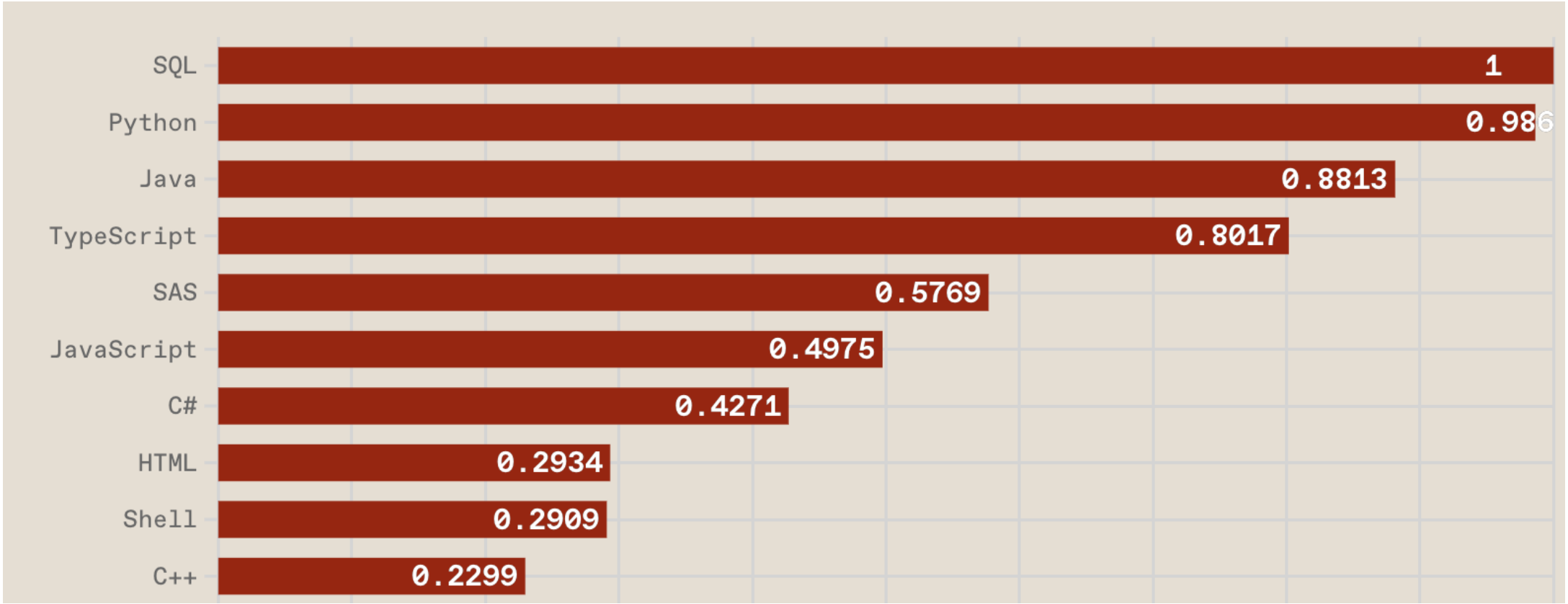
BYU

INFORMATION STRUCTURING IN DATABASES

- **Schema**: structure and organization of the database
- **Attribute**: relevant piece of information about the things of interest
- **Relationship**: connection between things of interest
- **Examples**:
  - **Customer Database**:
    - Attributes: Name, address, phone, account balance, credit rating, credit card number.
    - Relationships: Purchase and payment details.
  - **Student Database**:
    - **Attributes**: Name, address, phone, total/completed hours, GPA, standing.
    - **Relationships**: Courses, grades, library records, campus credit balance.

BYU

TOP PROGRAMMING LANGUAGES 2024 (BASED ON JOB POSTINGS)



Stephen Cass, 22 Aug 2024 https://spectrum.ieee.org/top-programming-languages-2024

BYU

STRUCTURED QUERY LANGUAGE (SQL)

- SQL was initially developed at in the early 1970s
- Initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's original database management system
- In the late 70's Oracle saw the potential and helped popularize SQL
  - 1979 – Oracle markets the first relational DB with SQL
- 1986 – ANSI SQL released
  - 1989, 1992, 1999, 2003 – Major ANSI standard updates
  - SQL 23 is the latest release

BYU

MANY DIALECTS OF SQL

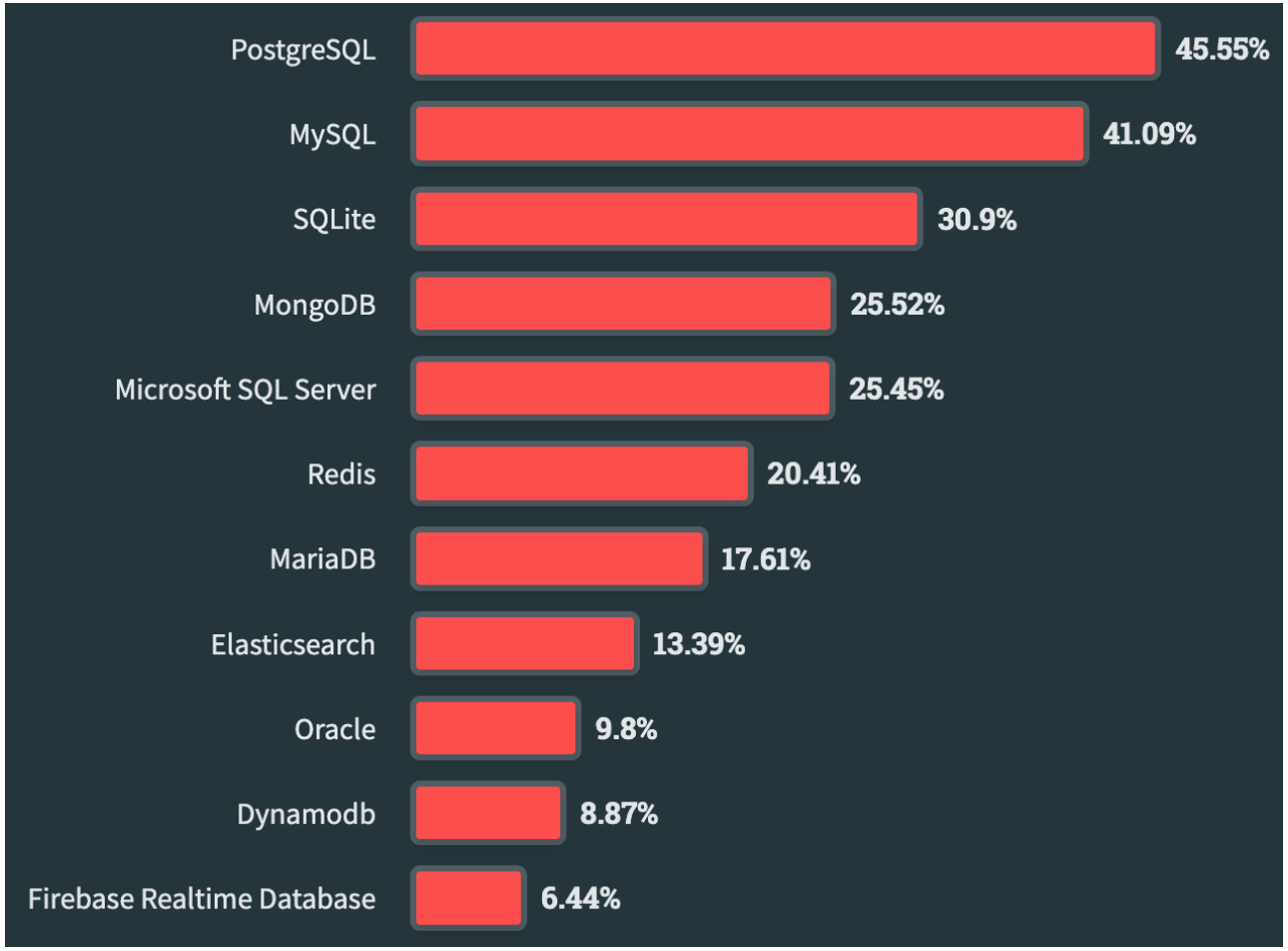
ClearScape Analytics

(T-SQL)

PL/SQL

BYU

POPULAR DATABASE MANAGEMENT SYSTEMS 2024

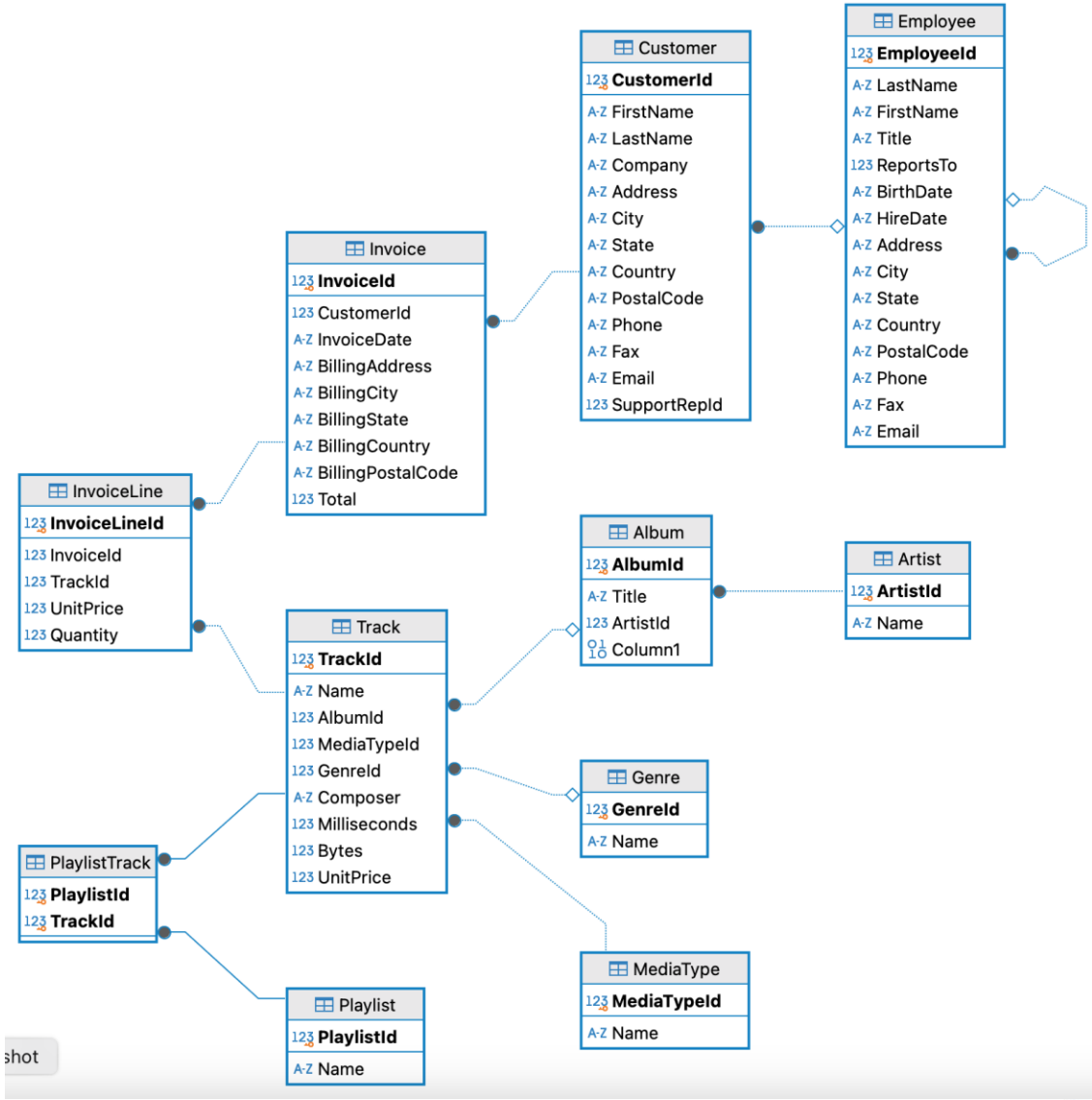


Stack Overflow Developer Survey, 2024

BYU

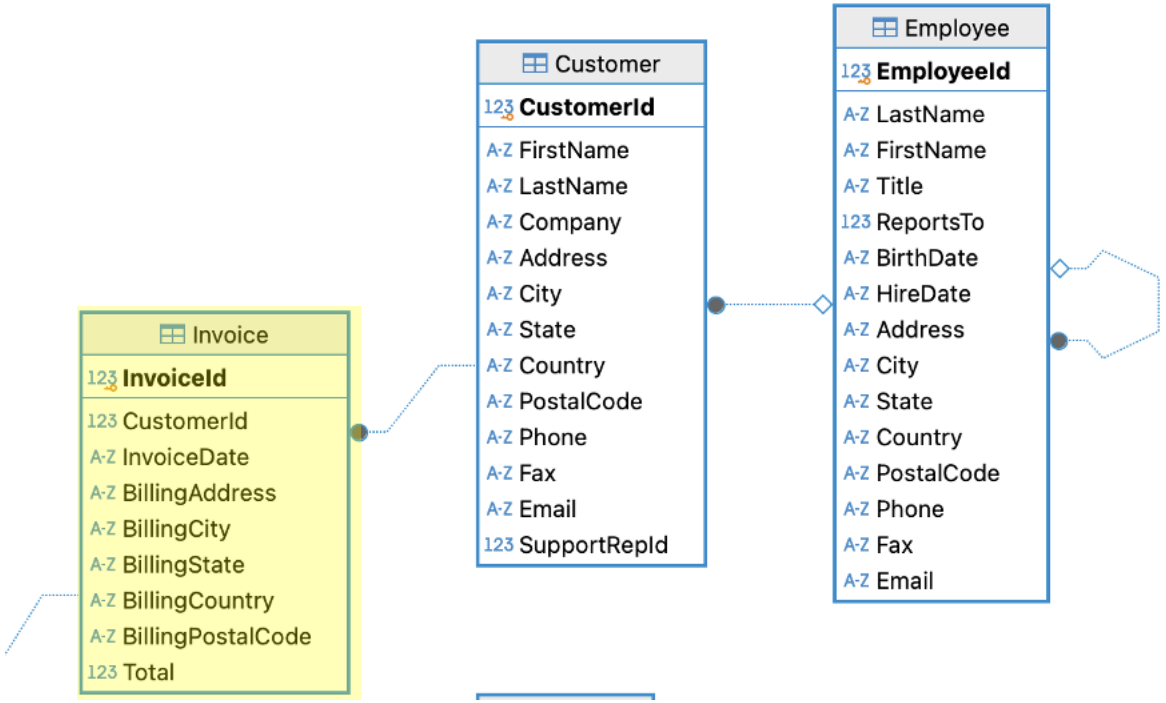
DATABASE SCHEMA

- The following is **Entity-Relationship Diagram (ERD)** and is used to summarize database schemas that create **Data Models**.
- This example represents a music store with key entities:
  - Customer,
  - Invoice,
  - InvoiceLine,
  - Track,
  - Album,
  - Artist,
  - Etc.



PRIMARY KEYS

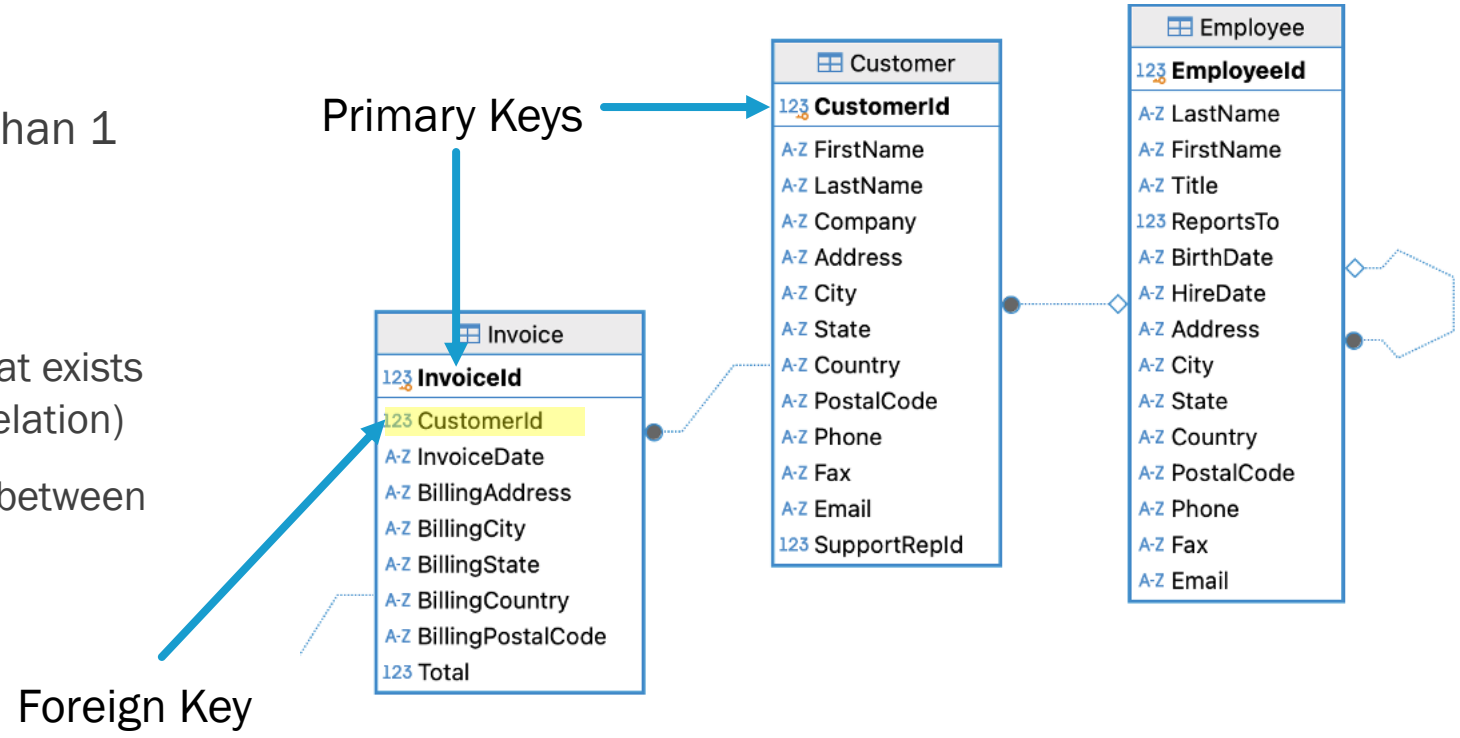
- Database rows (see **Invoice** Table to the right)
  - Contain information about instance **Invoice** (CustomerID, InvoiceDate, BillingAddress, etc.)
  - Each **Invoice** is represented by exactly 1 row
  - **Primary Key**: the key chosen as the attribute to uniquely define a row
  - Rows have identifiers to ensure that:
    - Data for each Invoice exists on only 1 row of the relation
    - Each row contains only data for 1 Invoice
    - **InvoiceID fills this role, thus each value of InvoiceID must be unique**



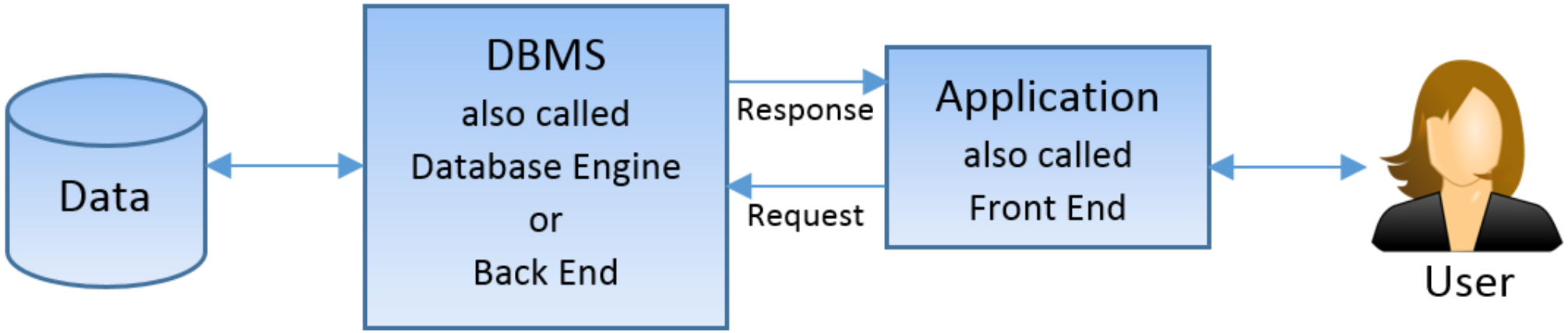
BYU

FOREIGN KEYS

- Notice that CustomerID exists in more than 1 one relation
- Foreign keys:
  - A primary key attribute from one relation that exists in another relation (can also be the same relation)
  - Used to define connections (*relationships*) between two entities (tables)



HIGH-LEVEL DATABASE APPLICATION ARCHITECTURE



BYU

WHAT IS A QUERY?

query

A query is a command for a database that typically inserts new data, retrieves data, updates data, or deletes data from a database.

query language

A query language is a computer programming language for writing database queries.

CRUD operations: *Create, Read, Update, and Delete data.*

BYU

SQL – STRUCTURED QUERY LANGUAGE

- Non-procedural language
  - Find the name of the customer with customer-id 192-83-7465

```
select   customer.customer-name
from     customer
where    customer.customer-id = '192-83-7465'
```

  - Find the balances of all accounts held by the customer with customer-id 192-83-7465

```
select   account.balance
from     depositor, account
where    depositor.customer-id = '192-83-7465' and
         depositor.account-number = account.account-number
```

- Application programs generally access databases through ether:
  - Language extensions to allow embedded SQL (we will show how to do this with python)
  - Application program interface (e.g. ODBC/JDBC) which allow SQL queries to be sent to a database

BYU

SELECT

- The **SELECT** statement is the most commonly used SQL command for retrieving data. It allows users to specify columns, source tables, and conditions for filtering results. The basic syntax is:

```
SELECT column1, column2, ...
FROM table_name
```

This query retrieves the first and last names of customers from the “Customer” table where the customers are in the USA.

Key Components:

1. **SELECT**: This keyword is followed by the column names (or * for all columns) you want to retrieve.
2. **FROM**: Specifies the table from which to retrieve the data.
3. **WHERE** (optional): Adds a condition to filter the rows based on specific criteria.
4. **ORDER BY**, **GROUP BY**, **HAVING** (optional): Additional clauses to organize, group, or filter aggregated data.

```
SELECT FirstName, LastName
FROM Customer
WHERE Country = 'USA';
```

BYU

WHERE

- The **WHERE** clause filters records in a query, retrieving only rows that meet specific conditions. It is applied after **FROM** and before optional clauses like **GROUP BY** or **ORDER BY**.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition:
```

Key Points:

- The **condition** in the WHERE clause filters data based on column values.
- It supports logical operators like AND, OR, and NOT for combining multiple conditions.
- It can work with comparison operators (=, !=, >, <, etc.), pattern matching (LIKE), ranges (BETWEEN), lists (IN), and more.

```
SELECT * FROM Customers
WHERE Country = 'USA' AND Age > 30;
```

This query selects all customers from the “Customers” table where the country is “USA”, and their age is greater than 30.

BYU

GROUP BY

- The **GROUP BY** clause groups rows with the same values into summary rows, often used with aggregate functions like **COUNT()**, **SUM()**, **AVG()**, **MIN()**, or **MAX()** to produce summarized results.

- **GROUP BY** groups the rows based on the unique values of one or more columns.
- It is typically used alongside aggregate functions to perform calculations on each group, such as counting occurrences or summing values.
- Every column in the SELECT statement that is not an aggregate function must be included in the GROUP BY clause.
- It can be followed by the HAVING clause to filter groups (similar to how WHERE filters rows).

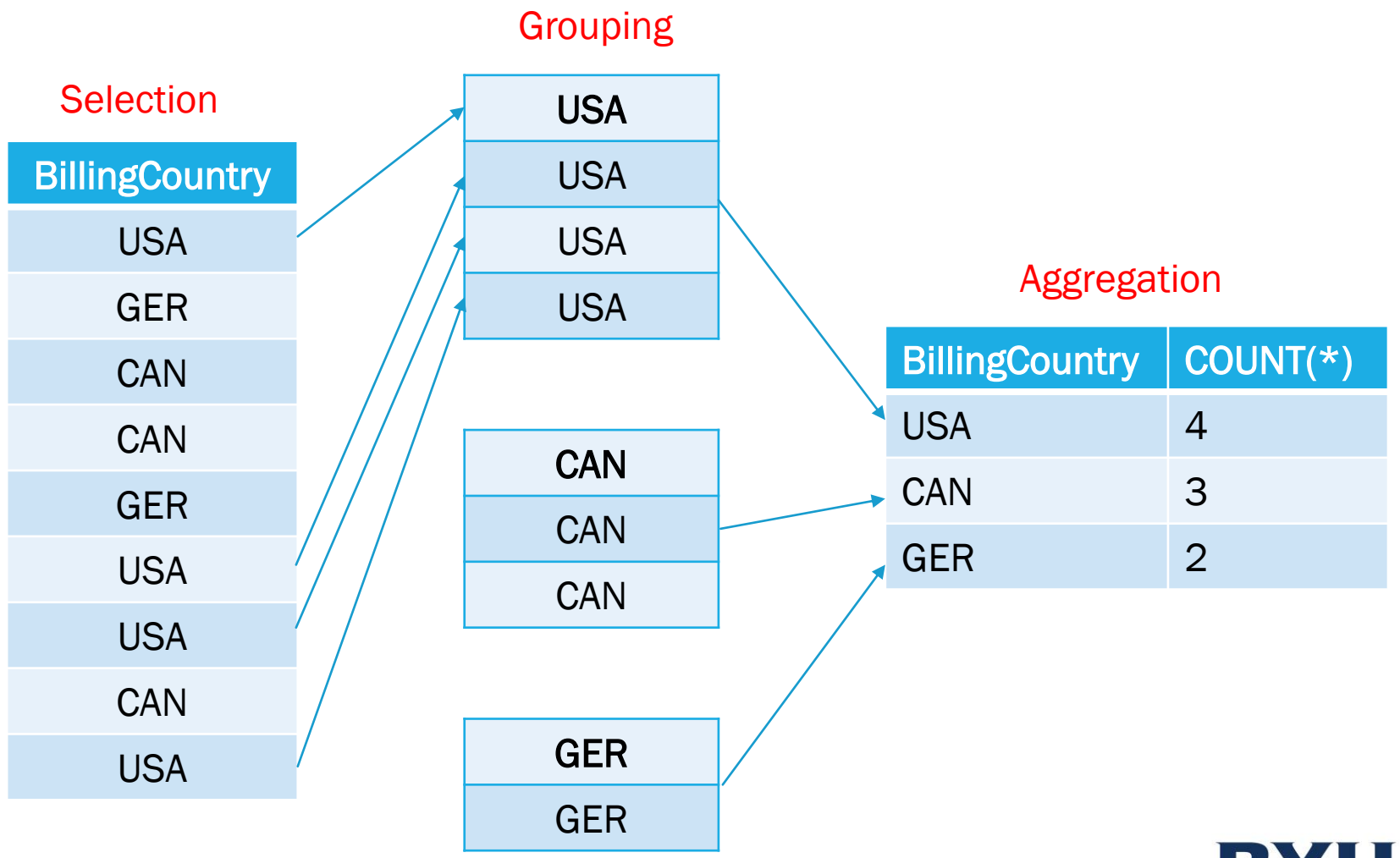
```
SELECT column1, aggregate_function(column2)
FROM table_name
WHERE condition
GROUP BY column1:
```

```
SELECT BillingCountry, COUNT(*)
FROM Invoice
GROUP BY BillingCountry;
```

BYU

GROUP BY ILLUSTRATED

```
SELECT BillingCountry, COUNT(*)
FROM Invoice
GROUP BY BillingCountry;
```



GROUP BY RESULTS

```
SELECT BillingCountry, COUNT(*)
FROM Invoice
GROUP BY BillingCountry;
```

A-Z BillingCount	123 COUNT(*)
Argentina	7
Australia	7
Austria	7
Belgium	7
Brazil	35
Canada	56
Chile	7
Czech Republic	14
Denmark	7
Finland	7
France	35
Germany	28
Hungary	7
India	13
Ireland	7
Italy	7

```
SELECT BillingCountry, COUNT(*)
FROM Invoice
WHERE BillingCountry IN ("USA",
"Canada","Germany")
GROUP BY BillingCountry;
```

A-Z BillingCountry	123 COUNT(*)
Canada	56
Germany	28
USA	91

BYU

HAVING

```sql
SELECT column1, aggregate_function(column2)
FROM table_name
WHERE condition
GROUP BY column1
HAVING aggregate_condition;
```

- The HAVING clause in SQL is used to filter the results of a GROUP BY query.
- It is similar to the WHERE clause, but the key difference is that **HAVING** operates on **aggregated** data **after** the **GROUP BY** operation, whereas **WHERE** filters rows **before** aggregation.

```sql
SELECT CustomerID, COUNT(InvoiceID) AS InvoiceCount
FROM Invoice
GROUP BY CustomerID
HAVING COUNT(InvoiceID) > 1;
```

- **GROUP BY**: Groups the invoices by CustomerID.
- **COUNT(InvoiceID)**: Counts the number of invoices for each customer.
- **HAVING COUNT(InvoiceID) > 1**: Filters only those customers who have more than one invoice.

BYU

ORDER BY

- The ORDER BY clause in SQL is used to sort the results of a query by one or more columns.
- You can specify whether the sorting should be in ascending (ASC) or descending (DESC) order.
- By default, the ORDER BY clause sorts in ascending order if not explicitly stated.
- ORDER BY is typically the **last clause** in a SQL query, after SELECT, FROM, WHERE, GROUP BY, and HAVING.
- Using ORDER BY can have performance implications, especially on large datasets, as the database needs to sort all the data before returning it.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC],
```

We want to retrieve the **CustomerID** and **Total spent** for all customers in the USA who have spent more than $1000 on invoices, and we'll group the data by customer and order by total spent.

```
SELECT CustomerID, SUM(Total) AS TotalSpent
FROM Invoice
WHERE Country = 'USA'
GROUP BY CustomerID
HAVING SUM(Total) > 1000
ORDER BY TotalSpent DESC;
```

1. **SELECT**: Retrieves the CustomerID and the total amount spent (SUM(Total)) for each customer.
2. **FROM**: Specifies the Invoice table where the data is stored.
3. **WHERE**: Filters for customers whose Country is the USA.
4. **GROUP BY**: Groups the results by CustomerID, so we can sum the Total for each customer.
5. **HAVING**: Filters the grouped data to include only customers whose total spending exceeds $1000.
6. **ORDER BY**: Orders the results by TotalSpent in descending order, so the highest spenders appear first.

BYU

COMMON AGGREGATION FUNCTIONS

```
--Find the total number of invoices in the Invoice table
SELECT COUNT(*) FROM Invoice;

--Calculate the total value of all invoices:
SELECT SUM(Total) FROM Invoice;

--Find the average total of invoices
SELECT AVG(Total) FROM Invoice;

--Find the maximum invoice total
SELECT MAX(Total) AS MaxInvoiceTotal FROM Invoice;

--Find the minimum invoice total
SELECT MIN(Total) AS MinInvoiceTotal FROM Invoice;
```

COUNT()
COUNT() counts the number of rows in the set.

MIN()
MIN() finds the minimum value in the set.

MAX()
MAX() finds the maximum value in the set.

SUM()
SUM() sums all the values in the set.

AVG()
AVG() computes the arithmetic mean of all the values in the set.

BYU

QUERY ORDER OR WRITING AND EXECUTION

SQL Clause	Order of Writing	Order of Execution
`SELECT`	1st	6th
`FROM`	2nd	1st
`JOIN`	3rd (if applicable)	2nd
`ON`	4th (if applicable)	3rd
`WHERE`	5th	4th
`GROUP BY`	6th	5th
`HAVING`	7th	7th
`ORDER BY`	8th	8th
`LIMIT`	9th	9th

BYU

CREATE TABLE <TABLE NAME>

- The CREATE TABLE statement in SQL is used to define a new table in a database.
- It specifies the table name, column names, their data types, constraints (e.g., primary key, foreign key), and any default values.

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    ...
);
```

```
CREATE TABLE VIPCustomer (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    JoinDate DATE DEFAULT CURRENT_DATE
);
```

Key Elements:

- **Table Name**: The name of the table being created.
- **Column Definitions**: The list of columns, each with a name, data type (such as INT, VARCHAR, DATE, etc.), and optional constraints (like PRIMARY KEY, NOT NULL, UNIQUE).
- **Constraints**: Define rules for the data in the table, such as:
- PRIMARY KEY: Uniquely identifies each record in the table.
- FOREIGN KEY: Links this table to another table.
- NOT NULL: Ensures a column cannot have a null value.
- DEFAULT: Provides a default value for a column.

BYU

CREATE TABLE <TABLE NAME> (WITH FOREIGN KEY)

If you wanted the VIPCustomer table to reference another table (for example, linking the CustomerID to a Customer table), you could add a **foreign key**.

We have two tables: VIPCustomer (that we created) and an existing table called Customer.

```
CREATE TABLE VIPCustomer (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    JoinDate DATE DEFAULT CURRENT_DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
```

We could have made it less confusing if we'd named the two IDs differently. ☺

In this case, the CustomerID column in the VIPCustomer table references the CustomerID column in the Customer table, establishing a link between the two tables.

BYU

INSERT INTO <TABLE>

- To insert data into the VIPCustomer table, you use the SQL INSERT INTO statement. The syntax depends on whether you're inserting values for all columns or just a subset of them

Example 1: Insert data into all columns:

```
INSERT INTO VIPCustomer (CustomerID, FirstName, Email)
VALUES (2, 'Jane', 'jane.doe@example.com');
```

In this example, values are provided for all columns in the VIPCustomer table. Make sure that the values match the data types specified for each column.

Example 2: Insert data into specific columns:

```
INSERT INTO VIPCustomer (CustomerID, FirstName, Email)
VALUES (2, 'Jane', 'jane.doe@example.com');
```

Here, only the CustomerID, FirstName, and Email are provided, so other columns will either take their default values or remain NULL if allowed

BYU

UPDATE <TABLE>

- To update existing records in a table, you use the SQL UPDATE statement along with the SET clause to specify which columns to modify
- The WHERE clause is critical in the UPDATE statement to target specific rows. If you omit it, **all rows** in the table will be updated.
- Always ensure that the values being inserted or updated conform to the data types and constraints defined in the table (e.g., not null, unique, etc.).

Example 1: Update a single column

```
UPDATE VIPCustomer
SET Email = 'john.newemail@example.com'
WHERE CustomerID = 1;
```

This updates the email of the customer with CustomerID = 1.

Example 2: Update multiple columns:

```
UPDATE VIPCustomer
SET LastName = 'Smith', Email = 'jane.smith@example.com'
WHERE CustomerID = 2;
```

Here, both the LastName and Email are updated for the customer with CustomerID = 2.

BYU

DROP TABLE <TABLE NAME>

- **DROP TABLE**: Completely deletes the table from the database.
- This action **cannot be undone**, so use it carefully.
- Any foreign key constraints that reference the table will also be affected, and depending on your database's configuration, you may need to drop those constraints first.

To delete the VIPCustomer table from the database, you would use the SQL DROP TABLE statement. This command removes the entire table, including all of its data, structure, and relationships.

```
DROP TABLE VIPCustomer;
```

If you want to keep the table structure but remove all the data, you would use the TRUNCATE statement instead:

```
TRUNCATE TABLE VIPCustomer;
```

BYU