|  |  |
| --- | --- |
|  | **Qatar University**  **College of Engineering**  **Department of Computer Science and Engineering** |

**Fall 2023**

**CMPS451**

**Database Management Systems**

**Course Project Report**

**phase 2**

**Submitted By**

|  |  |  |
| --- | --- | --- |
| **Name** | **QUID** | **Email** |
| Ridhwan Athaullah | 202005171 | [ra2005171@qu.edu.qa](mailto:ra2005171@qu.edu.qa) |
| Muhammad Muhaimin Mahdiyan | 201909592 | [mm1909592@qu.edu.qa](mailto:mm1909592@qu.edu.qa) |
| Mohamad Allaham | 202005872 | [ma2005872@qu.edu.qa](mailto:ma2005872@qu.edu.qa) |

|  |
| --- |
| **Table of Contents**  [**Front End** 3](#_Toc150774636)  [**Back End:** 3](#_Toc150774637)  [**Summary of Metadata** 4](#_Toc150774638)  [Metadata Table Columns 4](#_Toc150774639)  [Tables Metadata 4](#_Toc150774640)  [Column Information Metadata 5](#_Toc150774641)  [Index Metadata 6](#_Toc150774642)  [Cost Parameters, MySQL Screenshots, and Extra example 7](#_Toc150774643)  [**Type of Queries** 11](#_Toc150774644)  [**Query Testing** 11](#_Toc150774645) |

# **Front End**

For the frontend of the project, we opted to develop it as a mobile app using Kotlin and Android Studio. Users can navigate through the Cost Estimator and the Catalog of the tables. Additionally, users can input queries to display SELECT costs or utilize the drop-down menu to calculate the JOIN cost.

## Select Screen

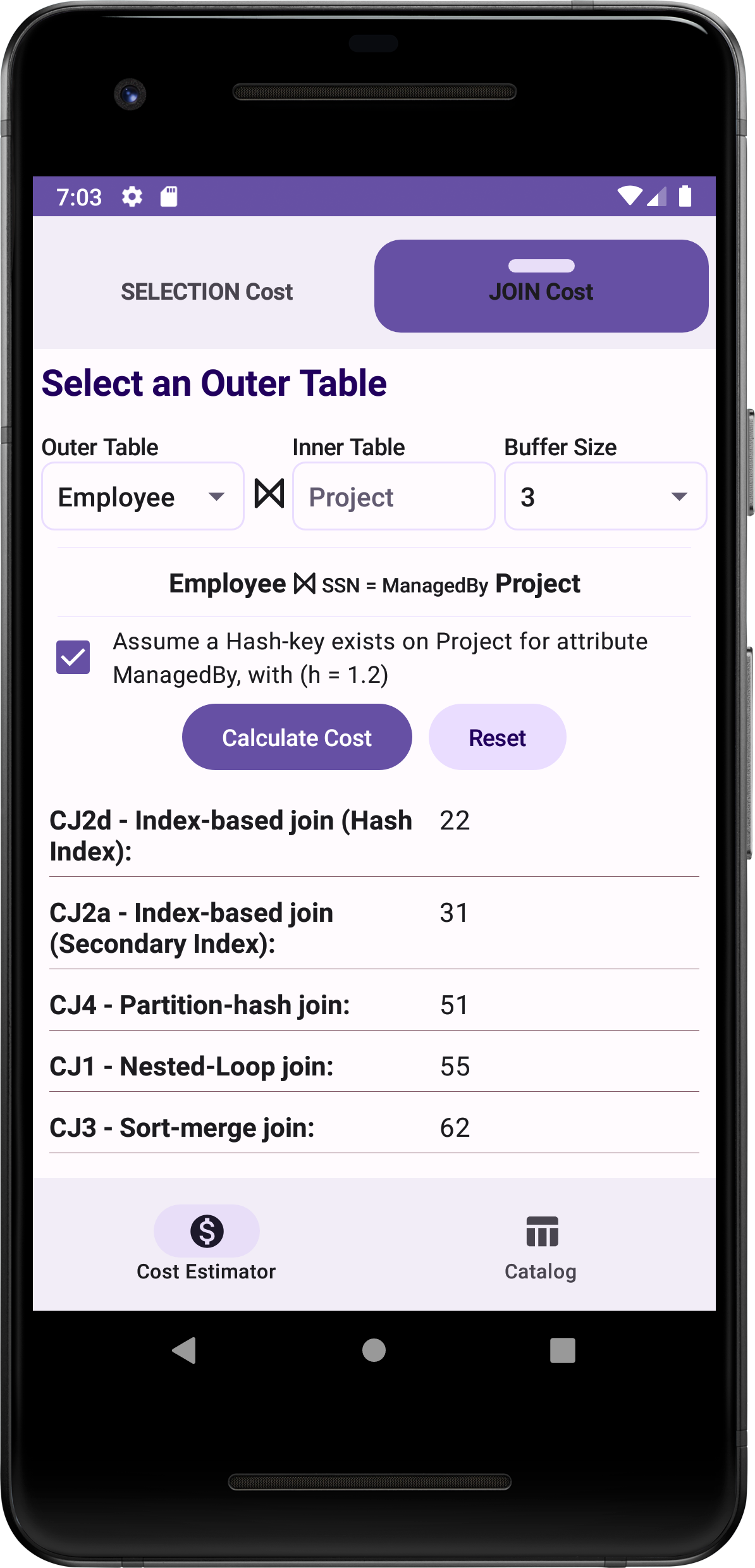
A screenshot of a phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

## Join Screen

A screen shot of a cell phone

Description automatically generated

## Catalog / Metadata Screen

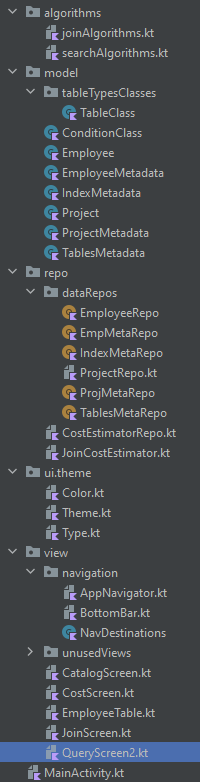
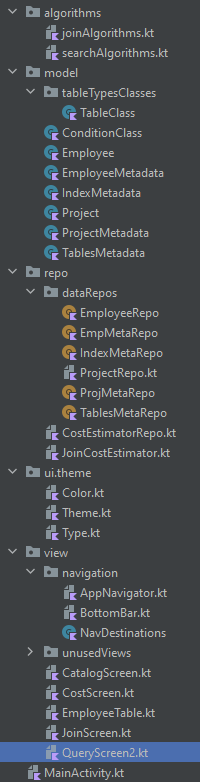
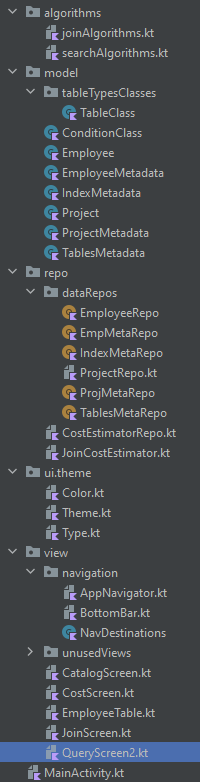
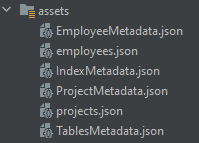
A screenshot of a phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

# **Back End:**

For the back end of our application, we have decided to go with the following file structure design and the previously mentioned Kotlin language which is similar to Java in some aspects.

Firstly, we decided to implement our app via the suggested second case where our cost-estimator is a stand-alone application and the all the data regarding our tables is stored locally as we personally felt it was easier to avoid using any API’s related to SQL database calls and are more experienced in using local files. The data starts of in the form of json files which is then converted into objects via their respective repos and classes located in the repo folder (for repos) and model folder (for classes).

We then implemented the different cost estimation algorithms within separate files in the algorithms folder depending on the type of operation they were used for (either join or select). These algorithms are then called as functions in different files.

The functions in the joinAlgorithms.kt file are called in the JoinCostEstimator.kt file where it is part of another function that generates a list of different costs for join related queries. This function is then called in the CostEstimatorRepo.kt and which is connected to our view for the join costs screen.

The functions in the searchAlgorithms.kt are called in another function that resides in the CostEstimatorRepo.kt which, similar to the previous case, creates a list of costs for the select queries and is called by our view related to calculating the cost of select queries. All 4 of the mentioned files have various utility functions within them to get the needed metadata value required for the calculations of the different costs.

To display the different values from the catalog/metadata we have decided to create a view that shows them as tables in the form of scrollable pictures to save time when compared to having to create a table composable/widget.

The remaining parts of the apps like the themes and the navigations are used to give our app its personality.

# **Summary of Metadata**

## Important Variables used in Cost sestimation

|  |  |  |
| --- | --- | --- |
| Varaible | Description/Meaning |  |
| **rowCount (r)** | Number of records (r) | Per Table |
| **blockCount (b)** | Number of blocks (b) |
| **Bfr** | Blocking factor (bfr) = num of records per block |
| **Selectivity (slA)** | Selectivity of attribute A (slA) | Per Attribute |
| **selectionCard (sA)** | Selection cardinality on attribute A (sA) = slA \* r |
| **indexLevels (xA)** | Number of levels in the index on attribute A |
| **firstLevelBlockCount (bl1A)** | Number of first level blocks of the index on attribute A |
| **NDV** | Number of Distinct Values “NDV (Attribute, Table)” |
| **Js** | Join selectivity (js) = 1 / max (NDV (A, R), NDV(B, S)) |
| **jc** | Join Cardinality (js) = js \* rR \* rS  rR = number of records in R, similarly for S |

## Tables Metadata

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tables Metadata (ASSUMED) ----------> Unit | | | | Bytes | Bytes | Block | Block | Bytes |
| **tableName** | **rowFormat** | **columnCount** | **rowCount** | **avgRowLength** | **dataLength** | **blockCount** | **bfr** | Width |
| EMPLOYEE | dynamic | 11 | 30 | 410 | 12300 | 13 | 2 | 380 |
| PROJECT | dynamic | 5 | 5 | 418 | 2090 | 3 | 1 | 418 |

* **Standard Block size** in MySQL DBMS is **1 Block** = **1 Kibibytes (KiB)** = **1024 Bytes**



* Data length in KiB = (avgRowLength \* rowCount) / 1024 Bytes

**dataLength** in Bytes = (avgRowLength \* rowCount)

* + Employee: 410 \* 30 = 12300
  + Project: 418 \* 5 = 2090
* **blockCount** = CEIL [ dataLength (Bytes) / Block Size (Bytes)] = CEIL [ dataLength / 1024 Bytes ]
  + Employee = 13
  + Project = 3
* **Blocking Factor** (**bfr**) = number of records per block = FLOOR [ rowCount / blockCount]
  + Employee = FLOOR [ 30 /13] = **2**
  + Project = FLOOR [ 5 / 3] = **1**
* **avgRowLength**: we will assume the average row length depending on the size of each data type/attribute in those records:  
    
    
    
    
  + **For average row length in EMPLOYEE Table:**

Table to shows the size of each data type in bytes:

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Size in Bytes** |
| SSN | varchar (9) | 9 |
| Fname | varchar (50) | 50 |
| Minit | char (1) | 1 |
| Lname | varchar (50) | 50 |
| DOB | date | 8 |
| Address | varchar (255) | 255 |
| Gender | enum ('Male’, ‘Female') | 14 |
| PhoneNo | varchar (15) | 15 |
| HireDate | date | 8 |
| Manager | tinyint (1) | 1 |
| ManagerSSN | varchar (9) | 9 |

As we can see, the size of a data type can vary depending on the type of data it is storing. Summing them up, to find the average row length we get: **410 Bytes**.

* + **For average row length in PROJECT Table:**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Size (bytes)** |
| ProjectNo | int | 4 |
| ProjectName | varchar(50) | 50 |
| Description | text | 255 |
| ProjectLoc | varchar(100) | 100 |
| ManagedBy | varchar(9) | 9 |

Summing the above we get an average row length of **418 Bytes**

* **Width:** refers to the size of the tuple (the sum of the sizes of each attribute) in bytes.

**Employee Width:**  9+50+1+50+8+255+14+15+8+1+9= **380 Bytes**

**Project Width:** 4+50+255+100+9= **418 Bytes**

## Column Information Metadata

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Employee Attribute | Type | Unique | indexName | Selectivity (sl) | selectionCardinality (s) | NDV | |
| SSN | varchar | true | Employee\_SSN | 0.033333333 | 1 | | 30 |
| Fname | varchar | false | Employee\_fname | 0.038461538 | 1.153846154 | | 26 |
| Minit | char | false | null | 0.111111111 | 3.333333333 | | 9 |
| Lname | varchar | false | Employee\_lname | 0.043478261 | 1.304347826 | | 23 |
| DOB | date | false | null | 0.037037037 | 1.111111111 | | 27 |
| Address | varchar | false | null | 0.045454545 | 1.363636364 | | 22 |
| Gender | enum | false | Employee\_gender | 0.5 | 15 | | 2 |
| PhoneNo | varchar | false | null | 0.066666667 | 2 | | 15 |
| HireDate | date | false | null | 0.037037037 | 1.111111111 | | 27 |
| Manager | tinyint | false | Employee\_manager | 0.5 | 15 | | 2 |
| ManagerSSN | varchar | false | Employee\_managerSSN | 0.333333333 | 10 | | 3 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Project Attribute | Type | Unique | indexName | Selectivity (sl) | selectionCardinality (s) | NDV |
| ProjectNo | int | true | Project\_projectNo | 0.2 | 1 | 5 |
| ProjectName | varchar | false | null | 0.2 | 1 | 5 |
| Description | text | false | null | 0.2 | 1 | 5 |
| ProjectLoc | varchar | false | null | 0.2 | 1 | 5 |
| ManagedBy | varchar | false | Project\_managedBy | 0.5 | 2.5 | 2 |

## 

## Index Metadata

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| indexName | Index | indexType | unique | level | selectivity | selectionCardinality | cardinality | firstLevelBlockCount |
| Employee\_SSN | primary | btree | true | 1 | 0.033333 | 1 | 30 | 3 |
| Employee\_managerSSN | secondary | btree | false | 1 | 0.25 | 7.5 | 4 | 1 |
| Employee\_fname | secondary | btree | false | 1 | 0.038461 | 1.153846 | 26 | 3 |
| Employee\_lname | secondary | btree | false | 1 | 0.043478 | 1.304348 | 23 | 3 |
| Employee\_gender | secondary | btree | false | 1 | 0.5 | 15 | 2 | 1 |
| Employee\_manager | secondary | btree | false | 1 | 0.5 | 15 | 2 | 1 |
| Project\_projectNo | primary | btree | true | 1 | 0.2 | 1 | 5 | 1 |
| Project\_managedBy | secondary | btree | false | 1 | 0.5 | 2.5 | 2 | 1 |

**\***btree represents a B+Tree storage organization.

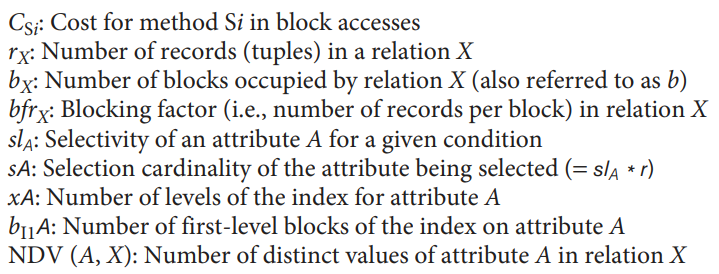
**\***The following indexes (**Employee\_SSN** - **Employee\_managerSSN** - **Project\_projectNo** - **Project\_managedBy**) were made by the innoDB, the storage manager of MySQL DBMS. The rest of the indexes were added by us for the soul reason of having multiple cost possibilities for certain queries like disjunction. In the disjunction, the cost will differ such that if all the involved attributes have an index, the cost will be less and if only one of the attributes does not have an index, the cost will be as high as when having no indexes on any of the attributes.

* Number of **first level blocks (**firstLevelBlockCount**) = CEIL [ num of index keys(cardinality) / fanout]:**

**Since we were not able to find an official source for MySQL B+-Tree fanout value, we assumed the fanout as 10.**

* + **Employee\_SSN:** CEIL [ 30 / 10] = **3**
  + **Employee\_managerSSN**: CEIL [ 4 / 10] = **1**
  + **Employee\_fname:** CEIL [ 26 / 10] = **3**
  + **Employee\_lname:** CEIL [ 23 / 10] = **3**
  + **Employee\_gender:** CEIL [ 2 / 10] = **1**
  + **Employee\_manager:** CEIL [ 2 / 10] = **1**
  + **Project**\_**projectNo**: CEIL [ 5 / 10] = **1**
  + **Project**\_**managedBy**: CEIL [ 2 / 10] = **1**
* **selectivity =** 1 / NDV(key, table)
* **selection cardinality (selectCard) =** table rowCount \* selectivity

## A text on a page Description automatically generatedCost Parameters, MySQL Screenshots, and Extra example



**Selection Cost Formulas  
A screenshot of a computer program

Description automatically generated**



A close up of a text

Description automatically generated

A close up of a text

Description automatically generated

# **Type of Queries**

## SELECT Query:

### Employee Table**:**

• **Primary** **Key** **and** Equality **Operator:**

**SELECT** **\*** **FROM** Employee **WHERE** SSN **=** '1234'**;**

**SELECT** **\*** **FROM** Employee **WHERE** SSN **=** '2345'**;**

• Non**-Primary** **Key** **on** Equality **Operator:**

**SELECT** **\*** **FROM** Employee **WHERE** Fname **=** 'Mia'**;**

**SELECT** **\*** **FROM** Employee **WHERE** Minit **=** 'R'**;**

**SELECT** **\*** **FROM** Employee **WHERE** Lname **=** 'Mia'**;**

**SELECT** **\*** **FROM** Employee **WHERE** DOB **=** '1988-06-14'**;**

**SELECT** **\*** **FROM** Employee **WHERE** Address **=** '456 Oak St'**;**

**SELECT** **\*** **FROM** Employee **WHERE** Gender **=** 'Female'**;**

**SELECT** **\*** **FROM** Employee **WHERE** PhoneNo **=** '555-123-4567'**;**

**SELECT** **\*** **FROM** Employee **WHERE** HireDate **=** '2017-09-05'**;**

**SELECT** **\*** **FROM** Employee **WHERE** Manager **=** **false;**

**SELECT** **\*** **FROM** Employee **WHERE** ManagerSSN **=** 1234**;**

• **Primary** **on** **Range** **Operator:**

**SELECT** **\*** **FROM** Employee **WHERE** SSN **>** 6781**;**

**SELECT** **\*** **FROM** Employee **WHERE** SSN **<** 6781**;**

**SELECT** **\*** **FROM** Employee **WHERE** SSN **>=** 6781**;**

**SELECT** **\*** **FROM** Employee **WHERE** SSN **<=** 6781**;**

• Non**-Primary** **Key** **using** **Range** **Operator:**

**SELECT** **\*** **FROM** Employee **WHERE** HireDate **>=** 'start\_date'

### Project Table**:**

• **Primary** **Key** **and** Equality **Operator:**

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectNo **=** 101**;**

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectNo **=** 103**;**

• Non**-Primary** **Key** **on** Equality **Operator:**

**SELECT** **\*** **FROM** PROJECT **WHERE** ManagedBy **=** 5678

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectName **=** 'Software Development'

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectLoc **=** 'Tech Park'

**SELECT** **\*** **FROM** PROJECT **WHERE** Description **=** 'Development of a new software application'

• **Primary** **on** **Range** **Operator:**

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectNo **<** 103**;**

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectNo **>** 103**;**

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectNo **<=** 103**;**

**SELECT** **\*** **FROM** PROJECT **WHERE** ProjectNo **>=** 103**;**

• Non**-Primary** **Key** **using** **Range** **Operator:**

**SELECT** **\*** **FROM** PROJECT **WHERE** ManagedBy **<** 5678

**SELECT** **\*** **FROM** PROJECT **WHERE** ManagedBy **>** 5678

**SELECT** **\*** **FROM** PROJECT **WHERE** ManagedBy **<=** 5678

**SELECT** **\*** **FROM** PROJECT **WHERE** ManagedBy **>=** 5678

## JOIN Query

# **Query Testing**