Data Manipulation Statements

- SELECT
- INSERT
- UPDATE
- DELETE
- Refer to documentation for others

The above keywords are used for the so-call "CRUD" operations on data:

Create, **R**ead, **U**pdate, **D**elete

Data Definition Statements

- CREATE
  - TABLE
  - VIEW
  - …
- ALTER
  - TABLE
  - VIEW
  - …
- DROP
  - TABLE
  - VIEW
  - …

Data Definition Statements

- Data definition statements manipulate database objects such as:
  - Tables
  - Views
  - Indexes
  - Triggers
  - Functions
  - Procedures

- MySQL documentation: https://dev.mysql.com/doc/refman/8.0/en/sql-data-definition-statements.html

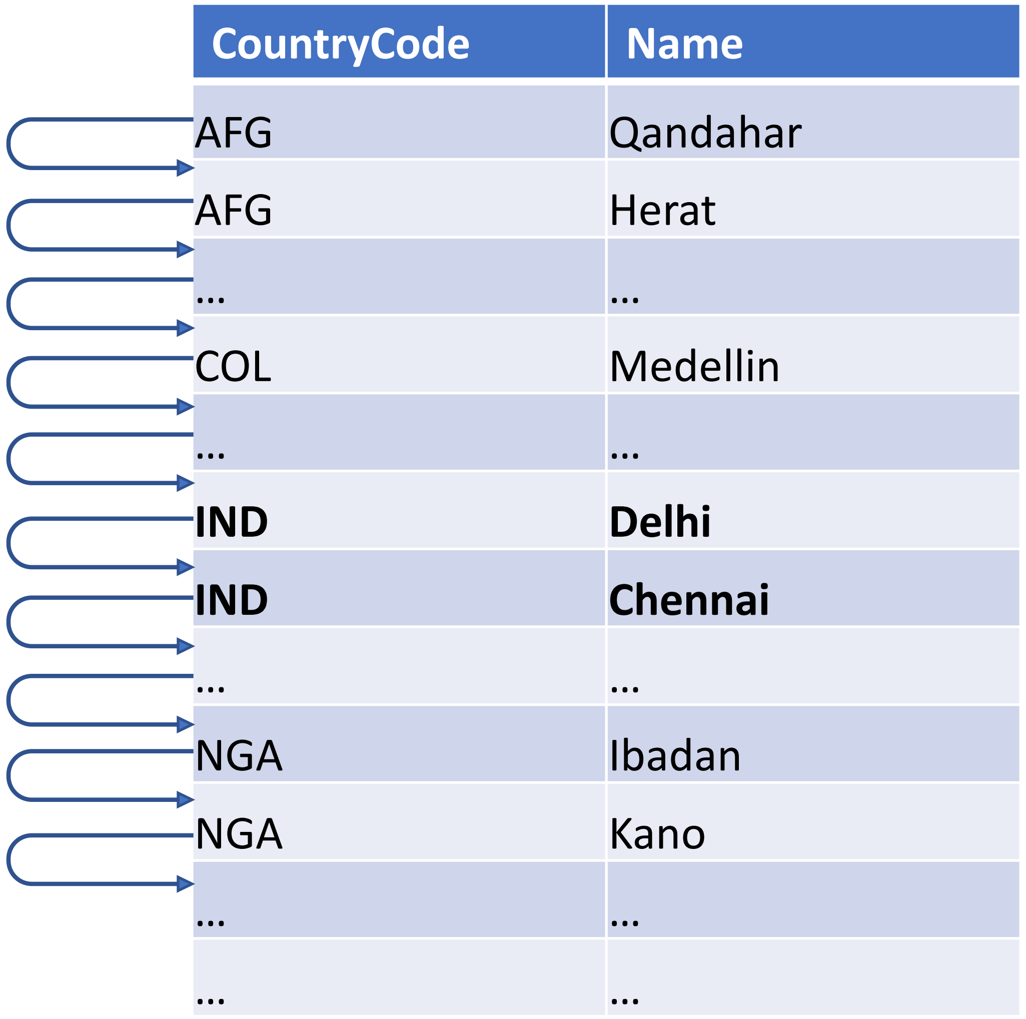
Indexes

- A structure associated with a table that can speed up data access.
- An index is defined on one or more columns on a table.
- Without an index, this statement would perform a **full table scan**:

```
SELECT * FROM city WHERE CountryCode = 'IND';
```

- **Full table scan**: A scan of table data in which the database sequentially reads all rows from a table and filters out those that do not meet the selection criteria.

world database
city full table scan



CountryCode	Name
AFG	Qandahar
AFG	Herat
…	…
COL	Medellin
…	…
IND	**Delhi**
IND	**Chennai**
…	…
NGA	Ibadan
NGA	Kano
…	…
…	…

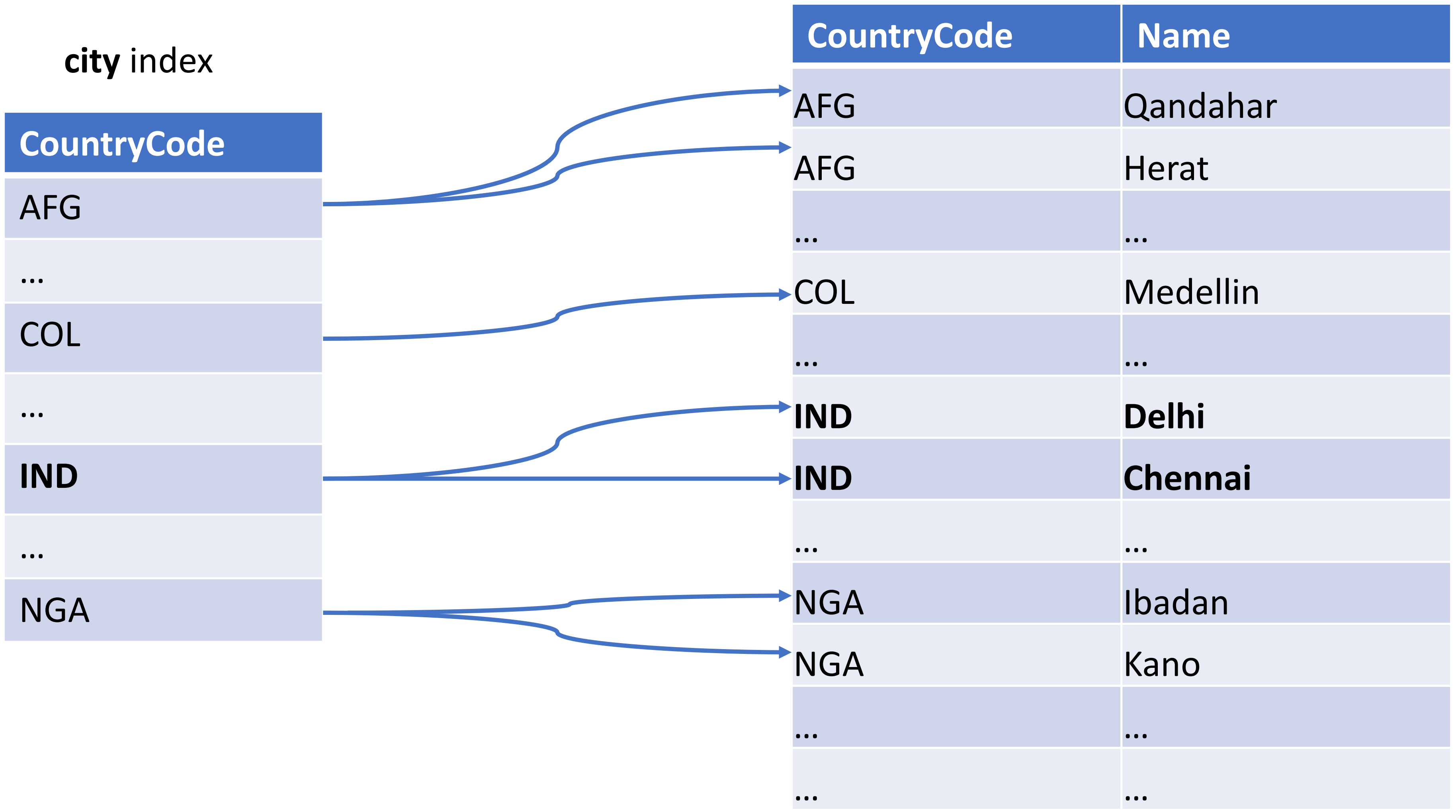
world database **city** table

```
CREATE TABLE `city` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  …
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
  CONSTRAINT `city_ibfk_1` FOREIGN KEY (`CountryCode`)
REFERENCES `country` (`Code`)
);
```

city search by index

city index

city table



CountryCode
AFG
…
COL
…
IND
…
NGA

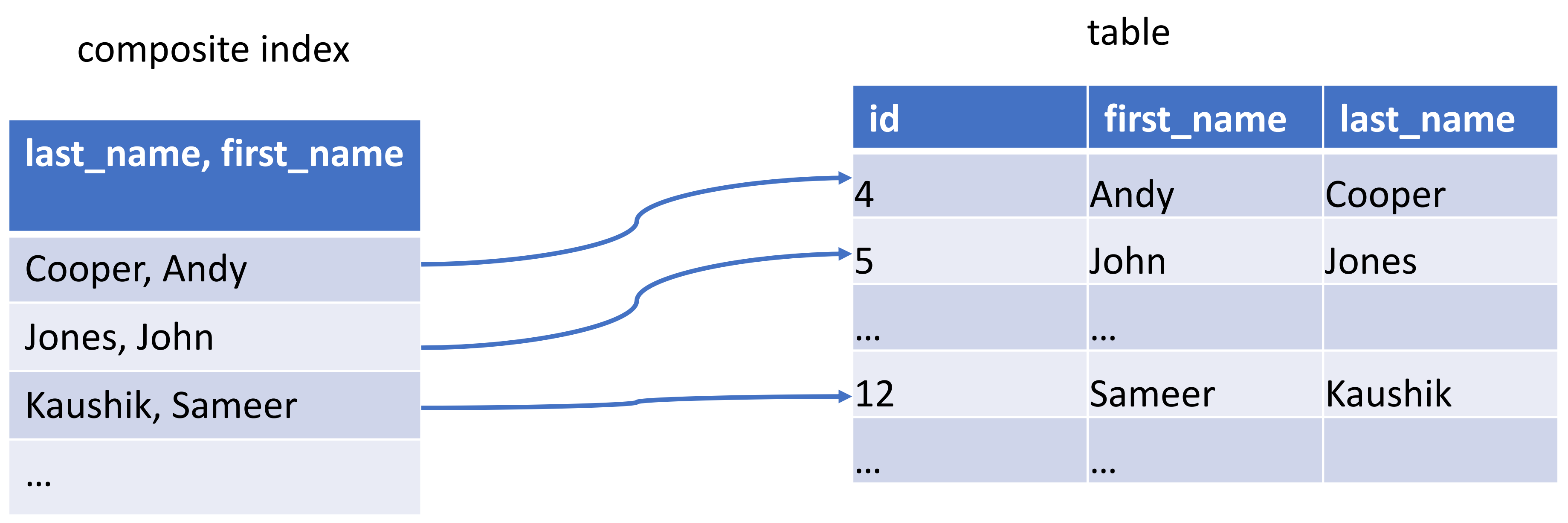
CountryCode	Name
AFG	Qandahar
AFG	Herat
…	…
COL	Medellin
…	…
IND	**Delhi**
IND	**Chennai**
…	…
NGA	Ibadan
NGA	Kano
…	…
…	…

Multiple column indexes

```
CREATE TABLE test (
      id INT NOT NULL,
      last_name CHAR(30) NOT NULL,
      first_name CHAR(30) NOT NULL,
      PRIMARY KEY (id),
      INDEX name (last_name,first_name)
);
```

The index can be used for lookups in queries that specify values in a known range for combinations of last_name and first_name values.

It can also be used for queries that specify just a last_name value because that column is a **leftmost prefix** of the index.



last_name, first_name
Cooper, Andy
Jones, John
Kaushik, Sameer
…

id	first_name	last_name
4	Andy	Cooper
5	John	Jones
…	…	
12	Sameer	Kaushik
…	…	

Queries using index:

```
SELECT * FROM test WHERE last_name='Jones';
SELECT * FROM test WHERE last_name='Jones' AND first_name='John';
```

Queries that will not use index:

```
SELECT * FROM test WHERE first_name='John';
SELECT * FROM test WHERE last_name='Jones' OR first_name='John';
```

Indexes

Indexes are used to find rows with specific column values quickly. They are defined on one or more columns.

MySQL Index keywords:

- PRIMARY KEY
- UNIQUE
- INDEX
- FULLTEXT: for text-based columns

Primary and foreign keys have indexes, by default.

Indexes

Benefits:

- Find rows with specific values quickly
- Avoid full table scans
- Eliminates rows from consideration
- Create indexes on foreign key columns to makes joins more efficient
- Sorting

B-tree

A tree data structure that is popular for use in database indexes.

- The structure is kept sorted at all times, enabling fast lookup for exact matches (equals operator) and ranges (for example, greater than, less than, and BETWEEN operators).

B-tree index

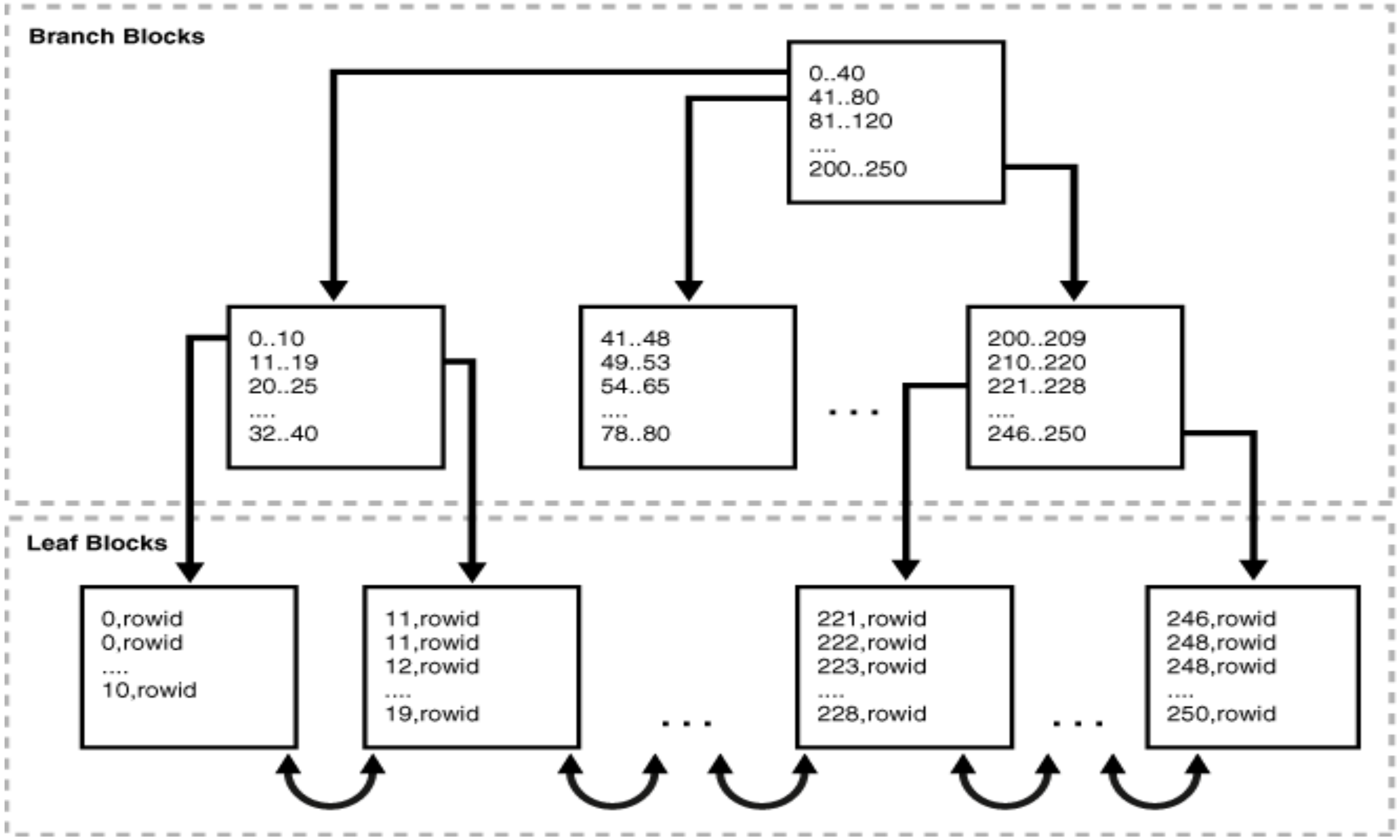


Image source: https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CNCPT721

Definitions

- rowid: A unique address for a row in a database
- Branch blocks: Used for searching for values
- Leaf block: Stores the indexed value and *rowid* to locate the actual row in the referenced table.
- A B-tree index is balanced: all leaf indexes have the same depth. Retrieval of any record in the index takes approximately the same time.

Index tradeoffs

- Indexes take up extra space.
- When an index is defined on a table, inserts, updates and deletes take longer as both the table and the index are modified.
- Use indexes only when necessary to speed up queries!

MySQL avoids using indexes in these cases

- The table is so small it is faster to do a full table scan than a key (index) lookup.
- There are no usable restrictions in the ON or WHERE clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table.
- You are using a key with low cardinality (many rows match the key value) through another column.

View all indexes on 'sakila'

```
SELECT DISTINCT
    TABLE_NAME,
    INDEX_NAME
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = 'sakila';
```

Readings on indexes

- https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html
- https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html#btree-index-characteristics