Learning Objectives

- Assign appropriate table properties
- Assign appropriate column options
- Create a table
- Alter a table
- Empty a table
- Remove a table
- Understand and use indexes accurately
- Assign and use foreign keys
- Obtain table and index metadata

Creating a Table

- General syntax for creating a table

```
CREATE TABLE <table> (
<column name> <column type> [<column options>],
[<column name> <column type> [<column options>],…,]
[<index list>]
)[<table options>];
```

- Example

```
CREATE TABLE CountryLanguage (
CountryCode CHAR(3) NOT NULL,
Language CHAR(30) NOT NULL,
IsOfficial ENUM('True', 'False') NOT NULL DEFAULT 'False',
Percentage FLOAT(3,1) NOT NULL,
PRIMARY KEY(CountryCode, Language)
)ENGINE = MyISAM COMMENT='Lists Language Spoken';
```

Table Properties

- Add table options to CREATE TABLE statement
- Several options available
  - ENGINE
  - COMMENT
  - CHARACTER SET
  - COLLATE
- Example

```
CREATE TABLE CountryLanguage (
...
) ENGINE=MyISAM COMMENT='Lists Language Spoken' CHARSET
   utf8 COLLATE utf8_unicode_ci;
```

COLLATE can also be used in SELECT queries.

Column Options (1/2)

- Add column options to CREATE TABLE statement
- Several options available
  - NULL
  - NOT NULL
  - DEFAULT
  - AUTO_INCREMENT
- Constraints
  - Restrictions placed on one or more columns
  - Primary Key
  - Foreign Key
  - Unique

Column Options (2/2)

- Column options example

```
CREATE TABLE City (
  ID int(11) NOT NULL AUTO_INCREMENT,
  Name char(35) NOT NULL DEFAULT '',
  CountryCode char(3) NOT NULL DEFAULT '',
  District char(20) NOT NULL DEFAULT '',
  Population int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (ID)
) ENGINE=MyISAM CHARSET=latin1
```

SHOW CREATE TABLE

- Viewing the exact statement used to create a table
- Example

```
SHOW CREATE TABLE City\G
*********************** 1. row ***********************
       Table: City
Create Table: CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (#.## sec)
```

Creating Tables from Existing Tables (1/4)

- Two methods
  - `CREATE TABLE…SELECT`
  - `CREATE TABLE…LIKE`
- `CREATE TABLE…SELECT` will create a new table to fit and store the result set returned by the `SELECT`
- `CREATE TABLE LIKE` creates a structurally equivalent table (alas no foreign keys), but does not copy any data

Creating Tables from Existing Tables (2/4)

- `CREATE TABLE...AS SELECT` can create a table that is empty or non-empty, depending on what is returned by the `SELECT` part

```
CREATE TABLE CityCopy1 AS SELECT * FROM City;
```

- Create a table that contains a selection of the contents:

```
CREATE TABLE CityCopy2 AS SELECT * FROM City
 WHERE Population > 2000000;
```

- Create an empty copy of an existing table:

```
CREATE TABLE CityCopy3 AS SELECT * FROM City LIMIT 0;
```

- Create a table that contains only specific columns:

```
CREATE TABLE CityCopy4 AS SELECT col1,col2 FROM City;
```

Creating Tables from Existing Tables (3/4)

- LIKE examples

```
CREATE TABLE t
 (i INT NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (i))
 ENGINE = InnoDB;

CREATE TABLE copy1 SELECT * FROM t WHERE 0;

CREATE TABLE copy2 LIKE t;
```

Creating Tables from Existing Tables (4/4)

- LIKE examples *(continued)*

```
SHOW CREATE TABLE copy1\G;
*************************** 1. row ***************************
       Table: copy1
Create Table: CREATE TABLE `copy1` (
  `i` int(11) NOT NULL default '0'
) ENGINE=MyISAM DEFAULT CHARSET=latin1

SHOW CREATE TABLE copy2\G;
*************************** 1. row ***************************
       Table: copy2
Create Table: CREATE TABLE `copy2` (
  `i` int(11) NOT NULL auto_increment,
  PRIMARY KEY  (`i`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- Some attributes (foreign keys, datadir options, etc.) not copied with **LIKE**

Temporary Tables

- For temporary use
- Only visible from within the current session
- Exist only during the current session
- May use an existing non-temporary table's name
  - Non-temporary table will be 'masked' while the temporary table still exists
- Not visible through either **SHOW TABLE** or information_schema.TABLES
- Temporary table containing cities associated with Texas:

```
CREATE TEMPORARY TABLE Texas AS
 SELECT Name FROM City WHERE District='Texas';
```

Add a Column

- Use an ALTER TABLE statement *with* ADD
- Example

```
ALTER TABLE City ADD COLUMN LocalName VARCHAR(35) CHARACTER SET utf8
 NOT NULL DEFAULT '' COMMENT 'The local name of this City';
```

- Structure Change

```
DESCRIBE City;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| ID          | int(11)     | NO   | PRI | NULL    | auto_increment |
| Name        | char(35)    | NO   |     |         |                |
...
| Population  | int(11)     | NO   |     | 0       |                |
| LocalName   | varchar(35) | NO   |     |         |                |
+-------------+-------------+------+-----+---------+----------------+
```

Remove a Column

- Use an ALTER TABLE statement *with* DROP
- Example

```
ALTER TABLE City DROP COLUMN LocalName;
```

Modifying Columns

- Use an ALTER TABLE statement *with* MODIFY
- Example

```
ALTER TABLE City MODIFY ID BIGINT NOT NULL AUTO_INCREMENT;
```

Changing Columns

- Use an ALTER TABLE statement *with* CHANGE
- Example
  - To change LastName column from CHAR(30) to CHAR(40)

```
ALTER TABLE HeadOfState CHANGE LastName LastName CHAR(40) NOT NULL;
```

  - To change name to Surname as well

```
ALTER TABLE HeadOfState CHANGE LastName Surname CHAR(40) NOT NULL;
```

Renaming Tables

- Use an ALTER TABLE statement *with* RENAME
- Examples

```
ALTER TABLE t1 RENAME TO t2;
RENAME TABLE t1 TO t2;
RENAME TABLE t1 TO tmp, t2 TO t1, tmp TO t2;
```

The DROP TABLE Command

- Remove a table
- Full or empty table
- `IF EXISTS` to avoid error
- `DROP TEMPORARY TABLE`
- Examples:

```
DROP TABLE table1;
DROP TABLE IF EXISTS table1;
DROP TEMPORARY TABLE EU_Countries_TEMP;
```

DROP TABLE has no UNDO feature, so be cautious when deleting an entire table!

Foreign Keys

- Distinct concepts
  - Foreign keys
    - References between rows throughout the databases
  - Relationships
    - Foreign keys are used to implement relationships between rows of data
  - Foreign key constraints
    - Used to maintain foreign keys and to ensure the references are kept consistent
  - Referential integrity
    - Foreign key constraints are used to enforce referential integrity

Foreign Keys and Relationships

- The ***world*** database
  - A row in the Country table represents a real county
  - A row in the City table represents a real city
- Real world relationship
  - Countries contain cities, and countries and cities are related to one another through this containment relationship
  - Of all the cities that belong to a country, there is one 'special' city known as the capital
    - City A resides in a particular country B
    - Country B can have a capital C which is also a city in that country
    - It is likely that city A is different from city C
  - The result is two entirely independent relationships between cities and countries

Foreign Keys Represent Relationships

- The data must represent the real world
  - If real countries and cities are related to one another, then rows from the City table and the rows from the Country table must be likewise related
- Real world has a process to represent relationship
  - For the most part, all cities that have a border that is enclosed within the border of a country are cities that belong to that country
- Databases have a process to represent relationship
  - A foreign key is a collection of one or more columns that has a combination of values in common with that of another collection of columns, usually in another table
  - A symbolic relationship using column values are used as a reference to the related row

Foreign Keys in the World DB

- The City table has a CountryCode column
  - Foreign key that relates to the Code column in the Country table
  - The Code column from the Country table will connect to those rows in the City table that have an identical value in their CountryCode column
  - All the rows from the City table that have a CountryCode value that matches the Code value from that row from the Country table apparently belong to the country it represents
- The Country table contains a Capital column
  - May be used to find a row in the City table that has an identical value in its ID column
  - The row from the City table that has an identical value in its ID column apparently represents the city that is the capital of the country

Foreign Key Example (1/2)

- Particular row from the City table

```
SELECT ID, Name, CountryCode FROM City WHERE Name LIKE 'Helsinki %';
+------+-----------------------+-------------+
| ID   | Name                  | CountryCode |
+------+-----------------------+-------------+
| 3236 | Helsinki [Helsingfors] | FIN         |
+------+-----------------------+-------------+
```

- What country is associated with the CountryCode 'FIN'?

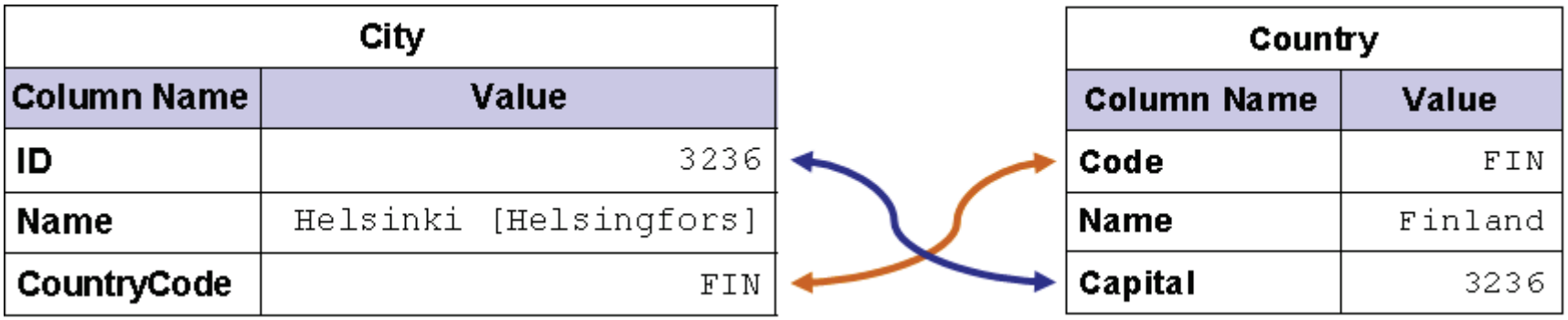
```
SELECT Code, Name, Capital FROM Country WHERE Code = 'FIN';
+------+---------+---------+
| Code | Name    | Capital |
+------+---------+---------+
| FIN  | Finland |    3236 |
+------+---------+---------+
```

- In which country is the city called 'Helsinki' situated?

```
SELECT ID, Name, CountryCode FROM City WHERE ID = 3236;
+------+-----------------------+-------------+
| ID   | Name                  | CountryCode |
+------+-----------------------+-------------+
| 3236 | Helsinki [Helsingfors] | FIN         |
+------+-----------------------+-------------+
```

Foreign Key Example (2/2)

- The following diagram illustrates these relationships and the foreign keys that implement them:



City	
Column Name	**Value**
ID	3236
Name	Helsinki [Helsingfors]
CountryCode	FIN

Country	
Column Name	**Value**
Code	FIN
Name	Finland
Capital	3236

Referential Integrity

- What would happen if the country code would change for a particular country?
  - Suppose the country code for Romania was changed from 'ROM' to 'ROU'

```
UPDATE Country SET Code = 'ROU' WHERE Code = 'ROM';
query OK, 1 row affected (#.## sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

  - Corresponding codes in City table unchanged
    - Those cities with a country code of ROM are no longer connected to a country
    - The referential integrity has been compromised
  - According to the data, the country of Romania has no cities
    - The database itself does not enforce referential integrity

Foreign Key Constraints

- A foreign key constraint will simply prevent a foreign key from referencing something that is not there
- Foreign key constraints take care of two things:
  - They prevent additions or changes to the referencing table that would result in a reference to something that does not exist
  - Changes to referenced rows can either be prevented or propagated to the referencing rows
    - A foreign key constraint can be defined in such a way that any change that will cause referential integrity problems are prevented from committing
    - A foreign key constraint can be defined to automatically propagate (cascade) any changes throughout the data to maintain referential integrity

Creating Foreign Key Constraints (1/3)

- Foreign keys constraints may be specified as part of the CREATE TABLE syntax

```
CREATE TABLE City (
 ID INT NOT NULL, Name CHAR(35) NOT NULL,
 CountryCode CHAR(3) NOT NULL, District CHAR(20) NOT NULL,
 Population INT NOT NULL, PRIMARY KEY (ID),
 FOREIGN KEY (CountryCode) REFERENCES Country (Code)
) ENGINE=InnoDB
```

- Alternatively they can be added to existing tables using an ALTER TABLE statement

```
ALTER TABLE City ADD FOREIGN KEY (CountryCode)
 REFERENCES Country (Code)
```

Creating Foreign Key Constraints (2/3)

- Full Foreign Key syntax

```
[CONSTRAINT [name]]
FOREIGN KEY [name] (referencing_col1[,..., referencing_colN])
REFERENCES referenced_tab (referenced_col1[,..., referenced_colN])
[ON DELETE {CASCADE | NO ACTION | RESTRICT | SET NULL}]
[ON UPDATE {CASCADE | NO ACTION | RESTRICT | SET NULL}]
```

- Mandatory elements
  - A list of referencing columns
  - The name of the referenced table
  - A list of referenced columns
  - The referenced columns should together form a PRIMARY KEY constraint

Creating Foreign Key Constraints (3/3)

- Optional elements
  - The constraint name
  - DELETE rule - specifies what should happen to the referencing rows in case a referenced row is removed
    - **CASCADE** means that the **DELETE** must be propagated to any referencing rows
    - **NO ACTION** means that a **DELETE** of a row from the referenced table must not occur if there are still referencing rows
    - **RESTRICT** means the same as **NO ACTION**
    - **SET NULL** means that the referencing columns in the referencing rows are changed to **NULL**
  - UPDATE rule - specifies what should happen to the referencing rows in case a referenced row is changed
    - Uses similar rules as those used for DELETE

Foreign Keys & Storage Engines (1/3)

- MySQL foreign keys are implemented at the storage engine level
  - The InnoDB engine is currently the only supported engine that provides a foreign key implementation

```
ALTER TABLE City ENGINE = InnoDB;
```

  - When attempting to create a foreign key on a non-InnoDB table, MySQL will silently ignore the request
    - Not even a a warning will be issued
  - When attempting to create a foreign key that references a non-InnoDB table, a runtime error occurs

```
ALTER TABLE City ADD CONSTRAINT fk_city_country
 FOREIGN KEY (CountryCode) REFERENCES Country(Code);
ERROR 1005 (HY000): Can't create table 'world.#sql-818_2' (errno: 150)
```

    - The error number 150 indicates some structural error in creating a foreign key constraint

Foreign Keys & Storage Engines (2/3)

- InnoDB engine status

```
SHOW ENGINE InnoDB STATUS;
...
----------------LATEST FOREIGN KEY ERROR-----------------
... Error in foreign key constraint of table world/#sql-818_2:
FOREIGN KEY bla (CountryCode) REFERENCES Country(Code):
Cannot resolve table name close to:
(Code)
...
```

  - InnoDB is looking for a table name near the occurrence of (Code) in the DDL statement, but can't seem to find one
    - The Country table is not a InnoDB table
    - InnoDB doesn't know anything about the existence of any non-InnoDB tables
    - The storage engine of the Country table would have to be changed to InnoDB before a foreign key constraint can be created

Foreign Keys & Storage Engines (3/3)

- InnoDB implementation of foreign keys
  - InnoDB requires an index to be present on the referencing columns
    - If such an index is not present already, one is automatically created
  - InnoDB requires the referenced columns to be the leftmost columns of some index defined on the referenced table
  - When changes are made to the data in either the referencing or the referenced tables, the foreign key constraint is checked in a row-by-row fashion
  - MySQL accepts the syntax for 'inline' foreign key constraints (foreign key constraint definitions at the column level)
    - However, they are silently discarded

Further Practice: Chapter 10

- Comprehensive exercises

Chapter Summary

- Assign appropriate table properties
- Assign appropriate column options
- Create a table
- Alter a table
- Empty a table
- Remove a table
- Understand and use Indexes accurately
- Assign and use foreign keys
- Obtain table and index metadata