



BOOTCAMP '20

SESSION #27

Session 26

- MySQL and Databases
- Installing and Testing MySQL
- How Shall We Store it? — Datatypes
- Designing and Creating Tables
- Populating the Database
- Retrieving the Data
- Changing Data
- Deleting Data
- Aggregate Functions
- Working with Dates and Times

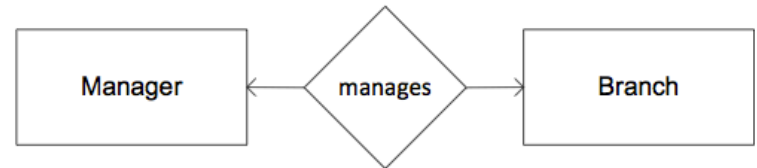
What will be covered?

- Relationships
- Indexes
- Joins
- Stored procedures
- Stored functions

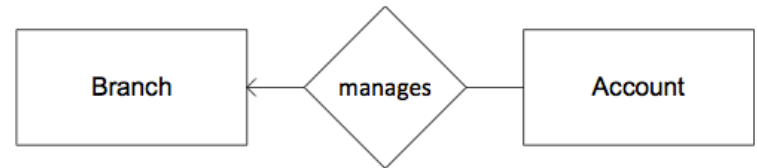
Relationships

- One-to-One
- One-to-Many
- Many-to-Many

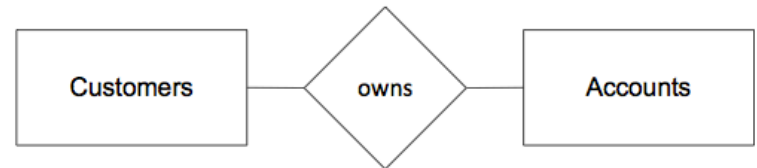
One-to-One



One-to-Many



Many-to-Many



Relationships

- A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.
- It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

- Syntax:

```
CREATE TABLE IF NOT EXISTS table_name (  
    ...  
    FOREIGN KEY column_name REFERENCES table_name (column_name);  
);  
  
ALTER TABLE table_name ADD FOREIGN KEY column_name REFERENCES table_name  
    (column_name);  
  
ALTER TABLE table_name DROP FOREIGN KEY column_name;
```

Relationships

- Possible ON DELETE and ON UPDATE are:
 - RESTRICT
 - CASCADE
 - SET NULL
 - NO ACTION (same as RESTRICT)
- Syntax:

FOREIGN KEY column_name **REFERENCES** table_name (column_name) **ON UPDATE** [RESTRICT| CASCADE|SET NULL|NO ACTION];

CONSTRAINT fk_name **FOREIGN KEY** column_name **REFERENCES** table_name (column_name) **ON UPDATE** [RESTRICT| CASCADE|SET NULL|NO ACTION];

Indexes

- A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

- Syntax:

```
CREATE TABLE IF NOT EXISTS table_name (
```

```
...
```

```
INDEX index_name (column_name);
```

```
INDEX index_name (column_name_1, ..., column_name_N);
```

```
);
```

```
ALTER TABLE table_name ADD INDEX index_name (column_name);
```

```
ALTER TABLE table_name ADD INDEX index_name (column_name_1, ..., column_name_N);
```

```
CREATE INDEX index_name ON table_name (column_name);
```

```
CREATE INDEX index_name ON table_name (column_name_1, ..., column_name_N);
```

```
DROP INDEX index_name ON table_name;
```

Indexes

What's this?

INDEX

- ABC, 164, 321ⁿ
academic journals, 262, 280–82
Adobe eBook Reader, 148–53
advertising, 36, 45–46, 127, 145–46, 167–68, 321ⁿ
Africa, medications for HIV patients in, 257–61
Agee, Michael, 223–24, 225
agricultural patents, 313ⁿ
Aibo robotic dog, 153–55, 156, 157, 160
AIDS medications, 257–60
air traffic, land ownership vs., 1–3
Akerlof, George, 232
Alben, Alex, 100–104, 105, 198–99, 295, 317ⁿ
alcohol prohibition, 200
Alice's Adventures in Wonderland (Carroll), 152–53
Anello, Douglas, 60
animated cartoons, 21–24
antiretroviral drugs, 257–61
Apple Corporation, 203, 264, 302
architecture, constraint effected through, 122, 123, 124, 318ⁿ
archive.org, 112
 see also Internet Archive
archives, digital, 108–15, 173, 222, 226–27
Aristotle, 150
Armstrong, Edwin Howard, 3–6, 184, 196
Arrow, Kenneth, 232
art, underground, 186
artists:
 publicity rights on images of, 317ⁿ
 recording industry payments to, 52, 58–59, 74, 195, 196–97, 199, 301, 329ⁿ–30ⁿ

Joins

INNER Join

Only returns connected, matching rows

martian.col1	martian.base_id	base.col2	base.base_id
🐉	1	🐉	1
🐉	2	🐉	2
🐉	3	🐉	3

RIGHT Join

Returns all connected rows, and unconnected rows from right table (nulls in left)

martian.col1	martian.base_id	base.col2	base.base_id
🐉	7	🐉	7
null	null	🐉	8
🐉	9	🐉	9

LEFT Join

Returns all connected rows, and unconnected rows from left table (nulls in right)

martian.col1	martian.base_id	base.col2	base.base_id
🐉	4	🐉	4
🐉	5	null	null
🐉	6	🐉	6

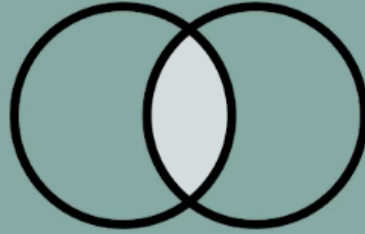
FULL Join

Returns connected rows & unconnected rows from both left & right tables

martian.col1	martian.base_id	base.col2	base.base_id
🐉	10	🐉	10
null	null	🐉	11
🐉	12	null	null

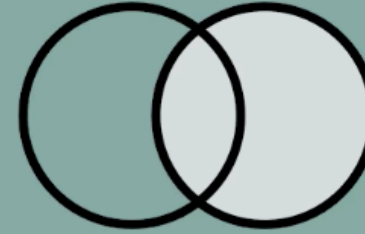
Joins

INNER Join



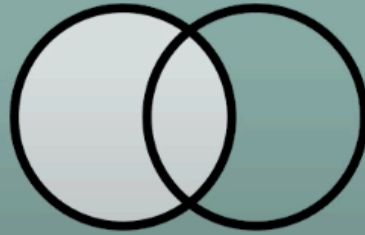
martian.col1	martian.base_id	base.col2	base.base_id
~~~~~	1	~~~~~	1
~~~~~	2	~~~~~	2
~~~~~	3	~~~~~	3

## RIGHT Join



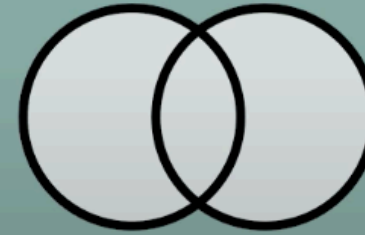
martian.col1	martian.base_id	base.col2	base.base_id
~~~~~	7	~~~~~	7
null	null	~~~~~	8
~~~~~	9	~~~~~	9

## LEFT Join



martian.col1	martian.base_id	base.col2	base.base_id
~~~~~	4	~~~~~	4
~~~~~	5	null	null
~~~~~	6	~~~~~	6

FULL Join



martian.col1	martian.base_id	base.col2	base.base_id
~~~~~	10	~~~~~	10
null	null	~~~~~	11
~~~~~	12	null	null

Stored procedures

- A procedure (often called a stored procedure) is a subroutine like a subprogram in a regular computing language, stored in database.
- A procedure has a name, a parameter list, and SQL statement(s).
- Almost all relational database system supports stored procedure, MySQL 5 introduce stored procedure.

Stored procedures

Here is the basic syntax of the `CREATE PROCEDURE` statement:

```
CREATE PROCEDURE procedure_name(parameter_list)
BEGIN
    statements;
END //
```

```
1  DELIMITER //
2
3  • CREATE PROCEDURE GetAllProducts()
4  BEGIN
5      SELECT * FROM products;
6  END //
7
8  DELIMITER ;
```

To execute a stored procedure, you use the `CALL` statement:

```
CALL stored_procedure_name(argument_list);
```

```
CALL GetAllProducts();
```

Read more: <https://www.w3resource.com/mysql/mysql-procedure.php>

Stored functions

- A stored function is a set of SQL statements that perform some operation and **return** a single **value**.
- Just like Mysql in-built function, it can be called from within a Mysql statement.
- By default, the stored function is associated with the default **database**.

```
CREATE FUNCTION `title_and_year`(title CHAR(200), year INT)
RETURNS char(205) DETERMINISTIC
BEGIN
    RETURN CONCAT(title, ' ', year);
END
```

QUESTIONS

