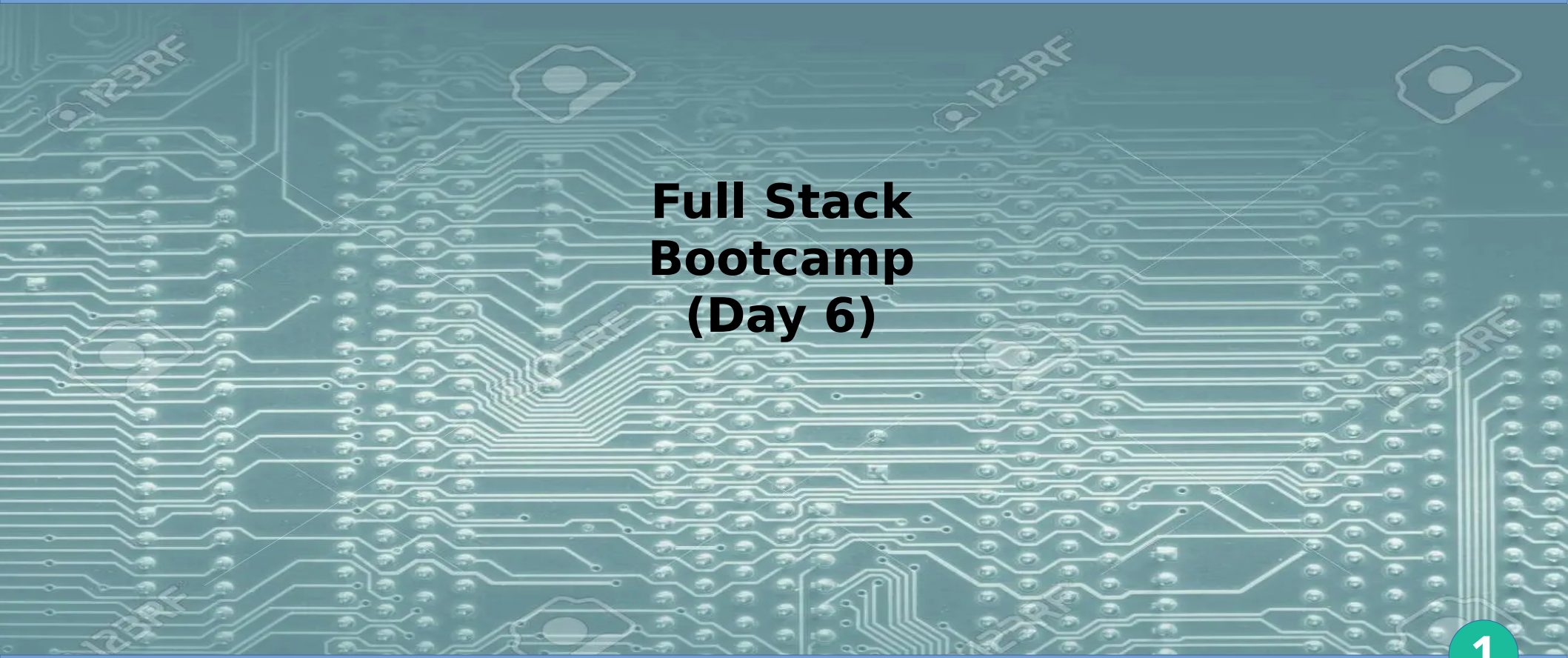
Relational databases Fundamentals



Full Stack Bootcamp (Day 6)

Agenda

- Day 1
  - ES6+
- Day 2
  - React Native
- Day 3
  - Angular
- Day 4
  - Springboot
  - SpringData
- Day 5
  - JSON
  - NoSQL
- Day 6
  - Relational
- Day 7
  - Junit
  - Mockito
- Day 8
  - Docker
- Day 9
  - Kubernetes
- Day 10
  - Images and tips

Review of Spring Data

- JPA
  - Entities
  - Repositories
  - Queries
  - Entity Managers
  - Services
  - MVC

Baseball statistics troubles

- Players are in different teams
- Players can change of team in the middle of the season
- Fielding and hitting statistics
- Pitching statistics
- Teams have different names and parks in their history
- You may want to know player salaries
- Other statistics

How relational databases works

- DB Engine
- Tables
- Columns
- Rows
- Store procedures

Pros

- DB Engines maturity
- Flexible
- Faster query processing
- Store Procedures (NO!, don’t!)
- Standardize language
- Portable

Cons

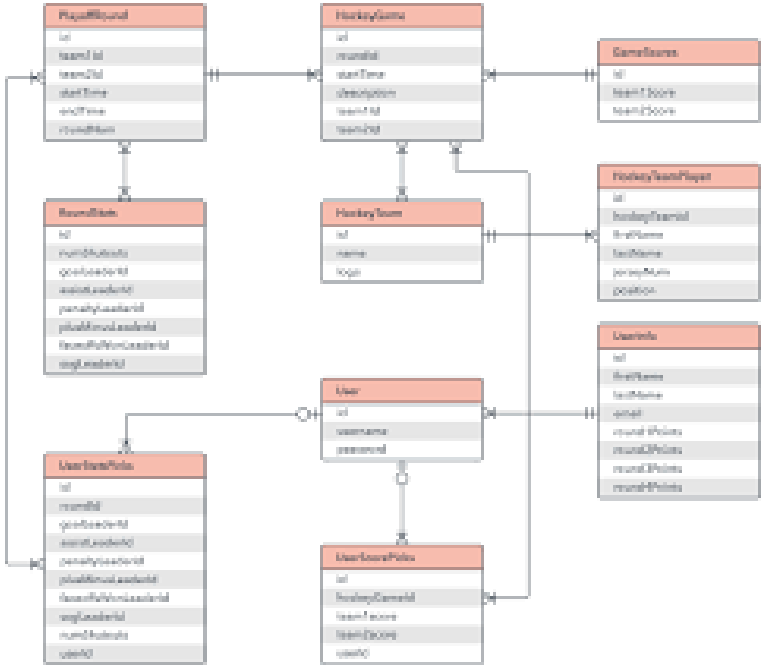
- Store Procedures (When bad used)
- Complex to design
- Complex to cluster o administer in comparison to NonSQL

Database engines

- Oracle
- DB2
- MySQL
- Postgresql
- SQLServer

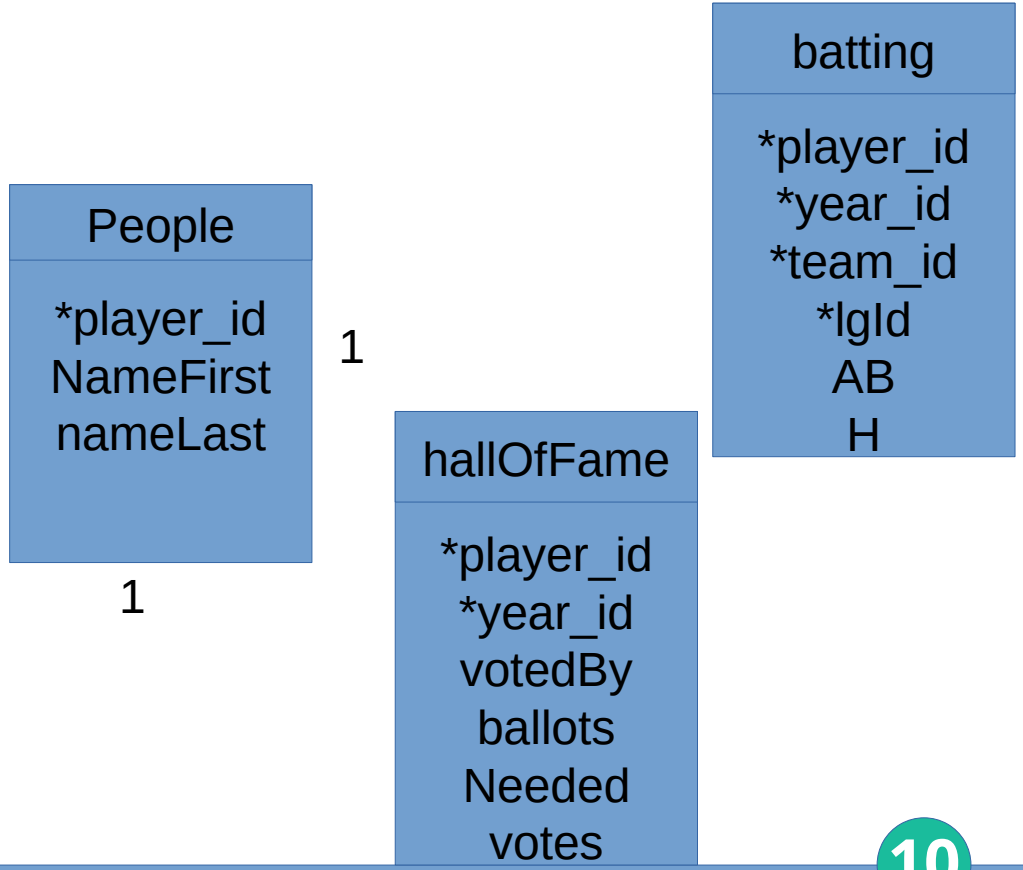
Analysis for Relational Dbs

- Schemas
- Entities and relations
- How Entities are found



Players and teams

- From previous days: Bating and players
- Baseball DataBank CSV
  - Teams
  - Teamsfranchises
  - Pitching
  - Halloffame



Importing data

- Varies from one dbengine to another
  - DB2
    - IMPORT FROM filename OF (IXF | ASC | DEL) INSERT INTO tablename
  - MySQL
    - LOAD DATA INFILE '/home/export_file.csv' INTO TABLE table_name FIELDS TERMINATED BY ',' ENCLOSED BY "" LINES TERMINATED BY '\n' IGNORE 1 ROWS;

Generating relationships

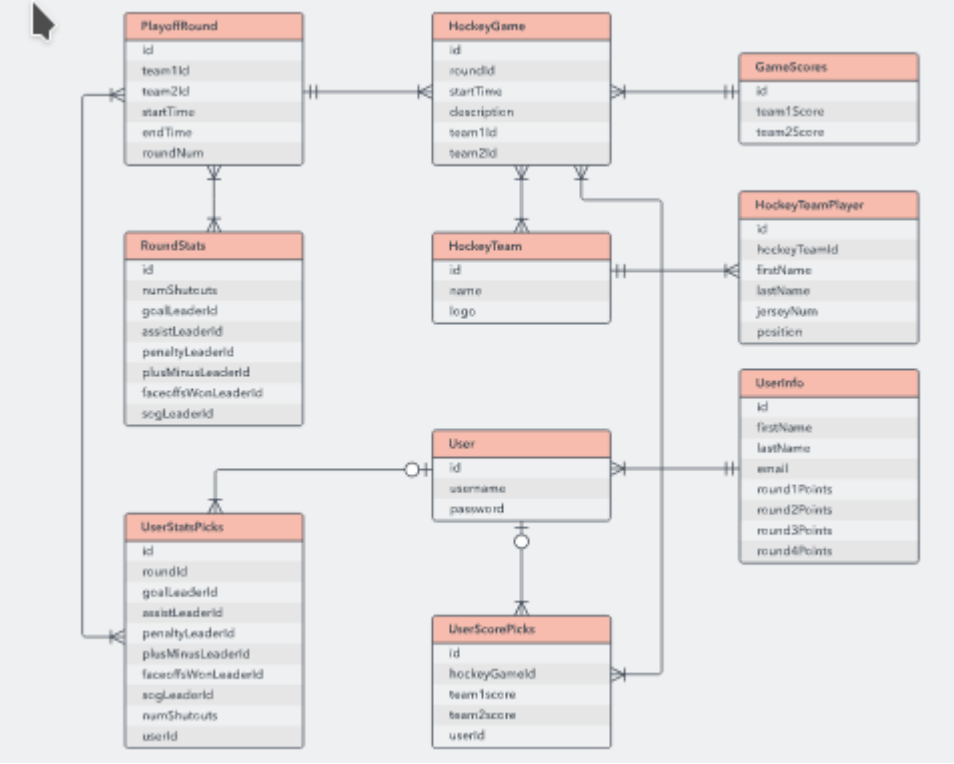
- Primary Keys
  - Unique values
  - Composite primary keys
- Foreign Keys

ERD

- An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

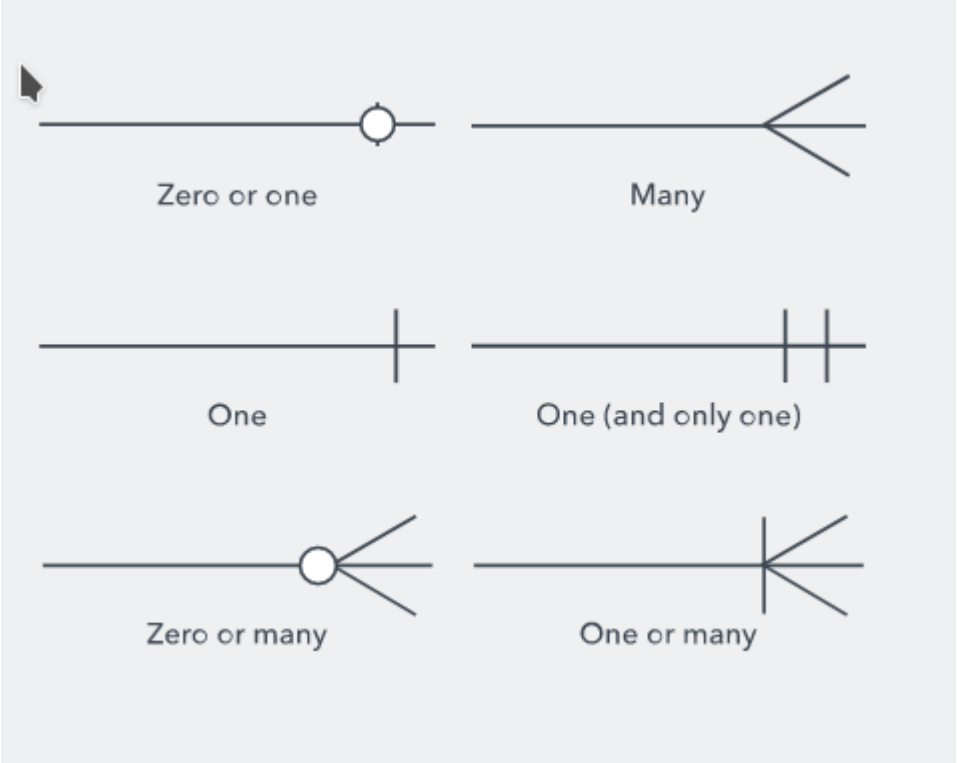
ERD

- Entities will be rows
- Attributes will be columns
- Lines in diagram means interactions
- One to many (Many to one)
- Key



ERD

- Cardinality



ERD and Data dictionaries

- Data Dictionary is a complete definition of a Database content
- Data Dictionary includes:
  - Tables definitions
  - Table description
  - Attributes definition
  - Attributes description
  - Primary keys
  - Foreign keys

Data types

- Numeric (Int, Long, SMALLINT, INTEGER, Decimal, Real, Double)
- Binary (BLOB)
- Character (CHAR, VARCHAR, CLOB)
- DATE, TIME, TIMESTAMP

Views

- View are representation of a selection that has values (Atributes) from several tables (Entities)
- Views are automatically updated when original tables
- Views don’t have keys

Indexes

- Queries by the key
- Selection performance
- Queries outside the keys
- Selecting with other tables

Command Types

- DDL (Data Definition language)
  - Create
  - Modify
  - Drop
- DML(Data Manipulation Language)
  - Select
  - Insert
  - Update
  - Delete
- DCL (Data Control Language)
  - Grant
  - Revoke

CREATE DATABASE

- CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
- [create_specification [, create_specification] ...]
- create_specification:
- [DEFAULT] CHARACTER SET charset_name
- | [DEFAULT] COLLATE collation_name

CREATE TABLE

- CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
- (create_definition,...)
- [table_options]
- [partition_options]

CREATE TABLE (2)

- create_definition: {
- col_name column_definition | {INDEX | KEY} [index_name] [index_type] (key_part,...)
- [index_option] ... | {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name] (key_part,...)
- [index_option] ... | [CONSTRAINT [symbol]] PRIMARY KEY
- [index_type] (key_part,...)
- [index_option] ... | [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
- [index_name] [index_type] (key_part,...)
- [index_option] ... | [CONSTRAINT [symbol]] FOREIGN KEY
- [index_name] (col_name,...)
- reference_definition | check_constraint_definition
- }

CREATE TABLE (3)

- column_definition: {
- data_type [NOT NULL | NULL] [DEFAULT {literal | (expr)}]
- [VISIBLE | INVISIBLE]
- [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
- [COMMENT 'string']
- [COLLATE collation_name]
- [COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
- [ENGINE_ATTRIBUTE [=] 'string']
- [SECONDARY_ENGINE_ATTRIBUTE [=] 'string']
- [STORAGE {DISK | MEMORY}

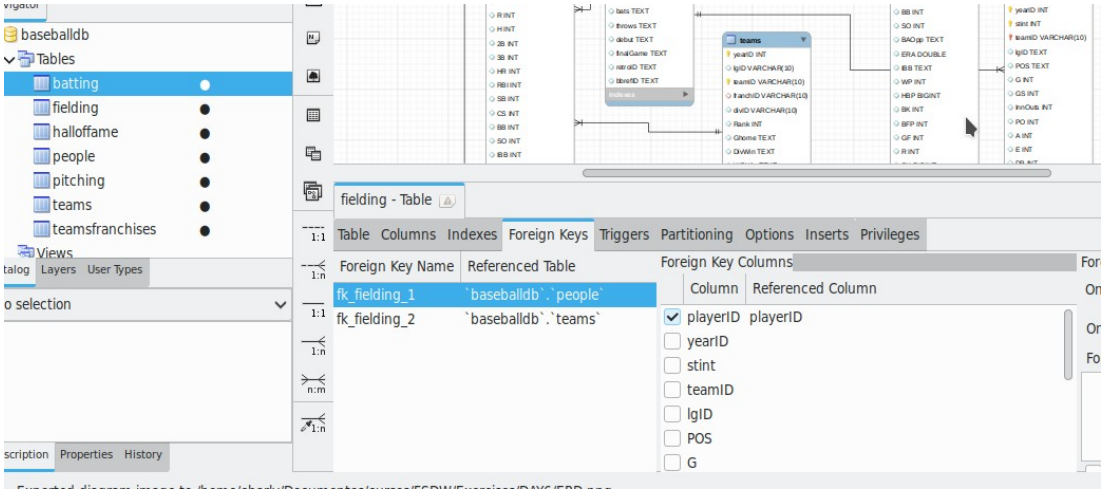
[reference_definition]
[check_constraint_definition]
| data_type
[COLLATE collation_name]
[GENERATED ALWAYS] AS (expr)
[VIRTUAL | STORED] [NOT NULL | NULL]
[VISIBLE | INVISIBLE]
[UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT 'string']
[reference_definition]
[check_constraint_definition]
}

CREATE TABLE (EXAMPLE)

```
CREATE TABLE pet (
        name VARCHAR(20),
        owner VARCHAR(20),
        species VARCHAR(20),
        sex CHAR(1),
        birth DATE,
        death DATE);
```

Tables

- mysql> SHOW TABLES;
- +---------------------+
- | Tables in menagerie |
- +---------------------+
- | pet |
- +---------------------+
- mysql> DESCRIBE pet;
- +---------+-------------+------+-----+---------+-------+
- | Field | Type | Null | Key | Default | Extra |
- +---------+-------------+------+-----+---------+-------+
- | name | varchar(20) | YES | | NULL | |
- | owner | varchar(20) | YES | | NULL | |
- | species | varchar(20) | YES | | NULL | |
- | sex | char(1) | YES | | NULL | |
- | birth | date | YES | | NULL | |
- | death | date | YES | | NULL | |
- +---------+-------------+------+-----+---------+-------+



Exercise 1

Import data

SQL SELECT From… Where

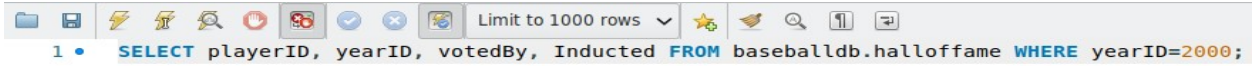
- Is the most common SQL command
  - Recovers records from DB
  - Joins the different tables
  - SQL is based on mathematical principles, specifically theory and relational algebra as well as set theory
  - Data is ordered as a SET of data records
  - Uses direct term of relational algebra as *PERMUTATION*, *PROJECTION*, *RESTRICTION* and *JOIN*

Retriving a complete table

- Use wild cards for selecting fields
- SELECT * FROM people;
- Using field (column) names:
- SELECT playerID, yearID, votedBy FROM baseballdb.halloffame;

Limiting rows

- The WHERE clause will let you limiting rows setting a condition of retrieval

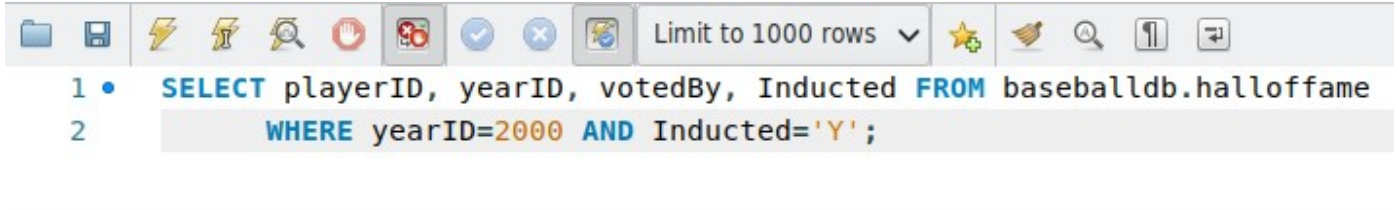


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	playerID	yearID	votedBy	Inducted
1	fiskca01	2000	BBWAA	Y
2	perezto01	2000	BBWAA	Y
3	riceji01	2000	BBWAA	N
4	cartega01	2000	BBWAA	N
5	suttebr01	2000	BBWAA	N
6	gossari01	2000	BBWAA	N
7	garvest01	2000	BBWAA	N
8	johnto01	2000	BBWAA	N
9	kaatji01	2000	BBWAA	N
10	murphda05	2000	BBWAA	N
11	morrija02	2000	BBWAA	N
12	parkeda01	2000	BBWAA	N
13	blylebe01	2000	BBWAA	N

Using logical operators

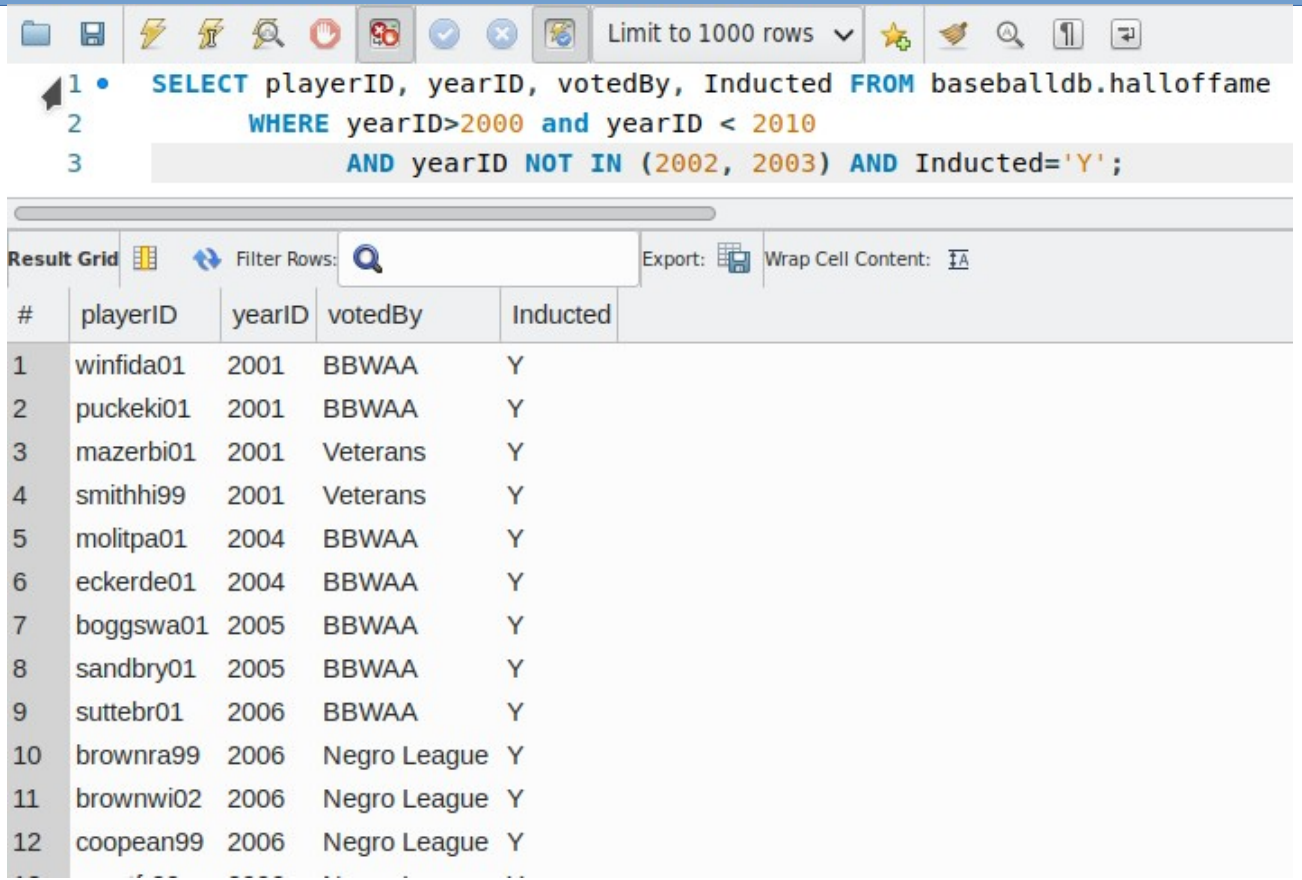
- You can use AND, OR, or IN clauses to set the conditions of WHERE



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	playerID	yearID	votedBy	Inducted
1	fiskca01	2000	BBWAA	Y
2	perezto01	2000	BBWAA	Y
3	andersp01	2000	Veterans	Y
4	mcphebi01	2000	Veterans	Y
5	steartu99	2000	Veterans	Y

NOT IN



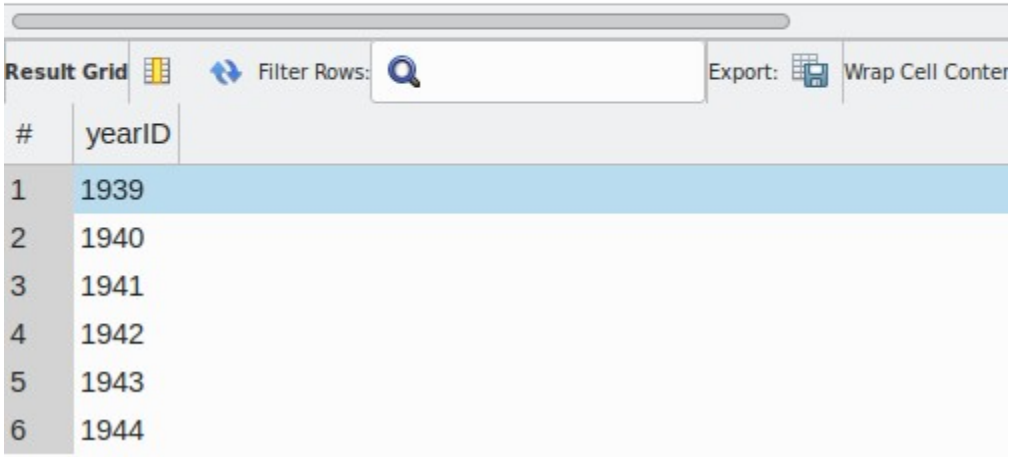
```
1 • SELECT playerID, yearID, votedBy, Inducted FROM baseballdb.halloffame
2       WHERE yearID>2000 and yearID < 2010
3           AND yearID NOT IN (2002, 2003) AND Inducted='Y';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	playerID	yearID	votedBy	Inducted
1	winfida01	2001	BBWAA	Y
2	puckeki01	2001	BBWAA	Y
3	mazerbi01	2001	Veterans	Y
4	smithhi99	2001	Veterans	Y
5	molitpa01	2004	BBWAA	Y
6	eckerde01	2004	BBWAA	Y
7	boggswa01	2005	BBWAA	Y
8	sandbry01	2005	BBWAA	Y
9	suttebr01	2006	BBWAA	Y
10	brownra99	2006	Negro League	Y
11	brownwi02	2006	Negro League	Y
12	coopean99	2006	Negro League	Y

DISTINCT

- Used for showing a value just once



#	yearID
1	1939
2	1940
3	1941
4	1942
5	1943
6	1944

DISTINCT

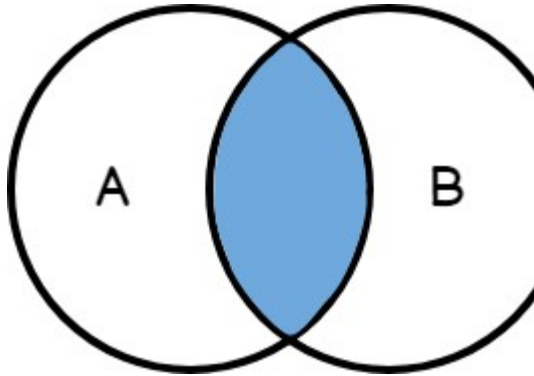
- Must be in the first columns of the query
- Column can have repeated values if forced by the next columns (for example in a query like SELECT DISTINCT yearID, playerID FROM baseballdb.batting WHERE yearID>1938 AND yearID<1945)

Order BY

- Help us to order our results depending a specific field.
  - The field to order by must be in the list of fields to show
  - If no ASC or DESC options ASC will be default

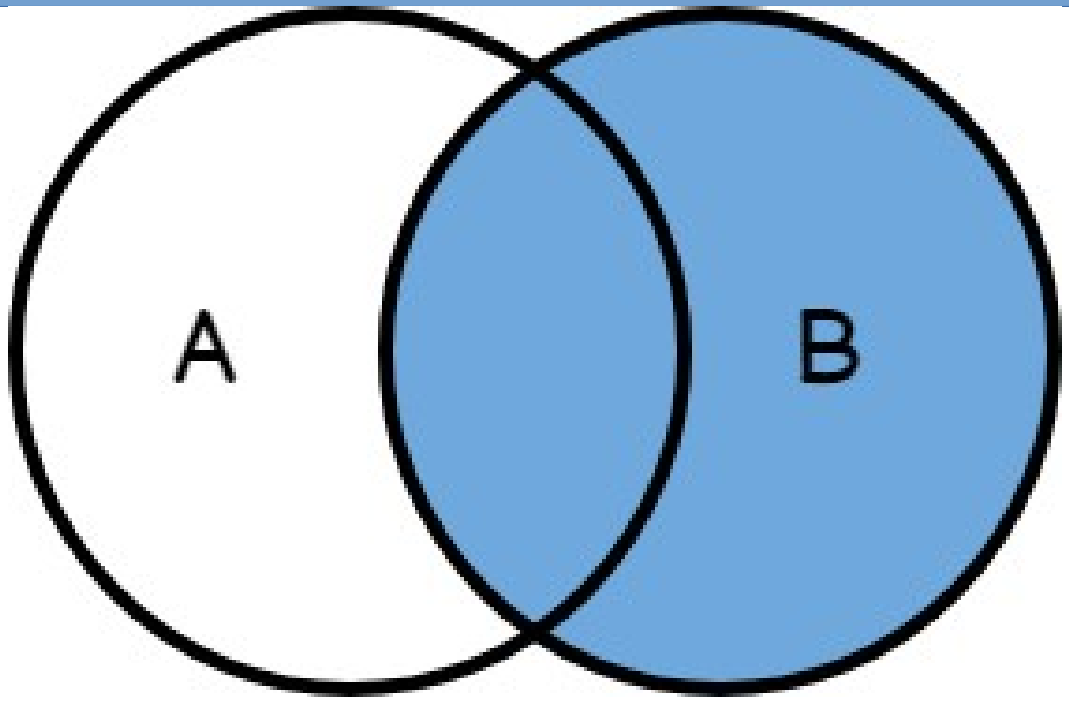
JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- If no join specified it will be acting as an Inner join
- Example: get all ORDERS FROM a specific customer
- SELECT p.Name AS ProductName,
- NonDiscountSales = (OrderQty * UnitPrice),
- Discounts = ((OrderQty * UnitPrice) * UnitPriceDiscount)
- FROM Production.Product AS p
- JOIN Sales.SalesOrderDetail AS sod
- ON p.ProductID = sod.ProductID
- ORDER BY ProductName DESC;



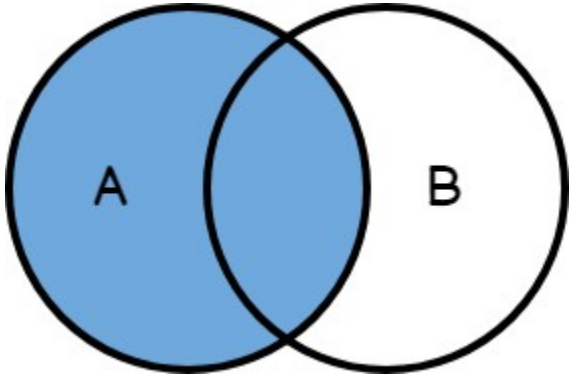
Right Join

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match



Left Join

- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.



Group by

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country"
- Group by is used often to create aggregations, for example “tell the total amount spent for each client all the year”
- **SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s) ORDER BY column_name(s)**;

SUM, AVG, COUNT

- SELECT COUNT(column_name)
- FROM table_name
- WHERE condition [GROUP BY column_name]
  - Count tells the number of records on the group
  - AVG returns the average of the value on the column for the group
  - SUM returns the sum of the value on the column for the group
  - Example: Get the average number of items that a client in NY bought last year

Nested Select commands

- Some times you need a set of values (or a simple value) to know the value of the real query: Get the list of the students with notes above the average:

```
SELECT * FROM students
WHERE GPA > (
    SELECT AVG(GPA)
    FROM students
);
```

Nested query, example 2

Let’s see how the **IN** operator works. In this example, you’ll calculate the average number of students in classes where the teacher teaches History or English:

```
SELECT AVG(number_of_students)
    FROM classes
    WHERE teacher_id IN (
        SELECT id    FROM teachers
        WHERE subject = 'English' OR subject = 'History');
```

CREATE VIEW

- Give the data of the player along with his personal information.
  - Common queries with different parameters.
  - **CREATE VIEW** `battingHist` **AS select** `batting`.`playerID` **AS** `playerId`,**sum**(`batting`.`G`) **AS** `G`,**sum**(`batting`.`AB`) **AS** `AB`,**sum**(`batting`.`H`) **AS** `H`,**sum**(`batting`.`2B`) **AS** `2B`,**sum**(`batting`.`3B`) **AS** `3B`,**sum**(`batting`.`HR`) **AS** `HR`,**sum**(`batting`.`RBI`) **AS** `RBI`,**sum**(`batting`.`SB`) **AS** `SB`,**sum**(`batting`.`CS`) **AS** `CS`,**sum**(`batting`.`BB`) **AS** `BB`,**sum**(`batting`.`SO`) **AS** `SO`,**sum**(`batting`.`IBB`) **AS** `IBB`,**sum**(`batting`.`HBP`) **AS** `HBP`,**sum**(`batting`.`SH`) **AS** `SH`,**sum**(`batting`.`SF`) **AS** `SF`,**sum**(`batting`.`GIDP`) **AS** `GIDP`,**avg**((`batting`.`H` / `batting`.`AB`)) **AS** `BAVG` **from** `batting` **group by** `batting`.`playerID`;
  - After the view is created can be used as a regular table
  - Get the summary og batting for “Babe Ruth”: **SELECT * FROM baseballdb.battingHist where playerId='ruthba01';**

EXERCISE 3

- ERD

SQL UPDATE

- **UPDATE** table_name
- **SET** column1 = value1, column2 = value2, ...
- **WHERE** condition;
- The update will be on the condition and that condition could be a Select statement, for example “Set all 2020 New York Yankees player to no hit in their batting statistics”
- The value can be an operation as well

SQL DELETE

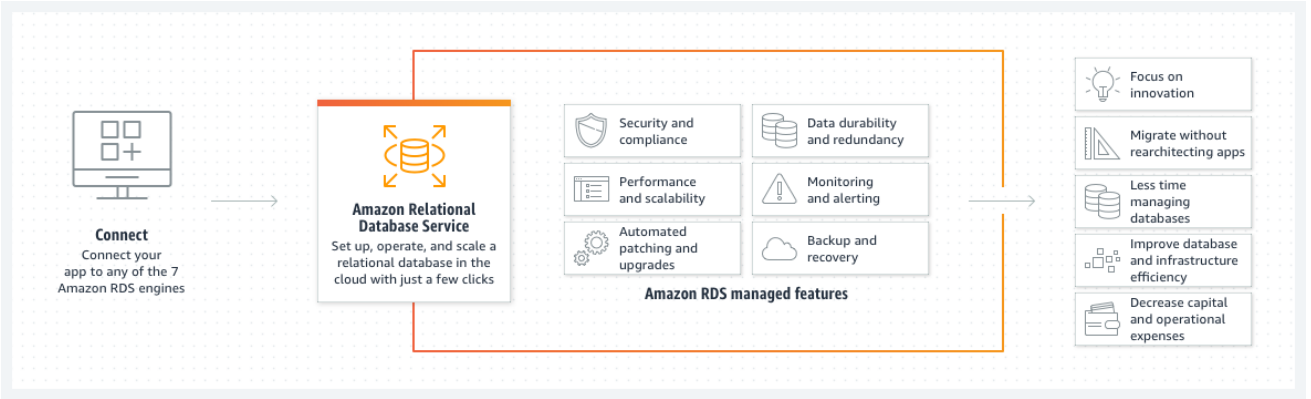
- **DELETE FROM table_name WHERE condition;**
- https://www.youtube.com/watch?v=i_cVJgIz_Cs&list=RDi_cVJgIz_Cs&start_radio=1 (Spanish)
- Cascade deletion
- Foreign keys and delete

EXERCISE 4

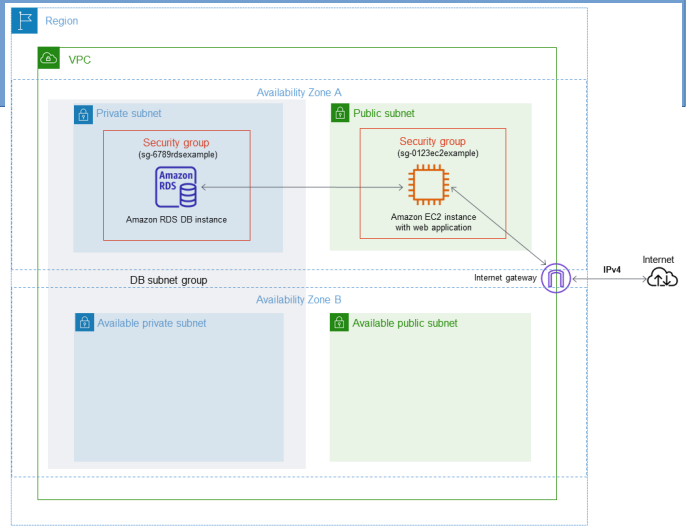
- Update and delete

AWS

- RBDS
  - MariaDB
  - Microsoft SQL Server
  - MySQL DB
  - Oracle
  - PostgreSQL



AWS Steps



- Get an AWS account and your root user credentials
- Create an IAM user
- Sign in as an IAM user
- Create IAM user access keys
- Determine requirements
- Provide access to your DB instance in your VPC by creating a security group
- Checkout this tutorial: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/TUT_WebAppWithRDS.html

GCP

- Cloud SQL
  - Fully managed relational database service for MySQL, PostgreSQL, and SQL Server. Run the same relational databases you know with their rich extension collections, configuration flags and developer ecosystem, but without the hassle of self management.

Day 6 summary

- SQL overview
- DB Engines
- Data type
- Use cases
- Cloud offerings