Introduction to SQL

IT TALENTS
Training Camp

Contents

1. Relational Databases and Data Models
2. SQL
3. Introducing SELECT Statement
    - The WHERE Clause
    - Sorting with ORDER BY
    - Selecting Data From Multiple Tables

Contents (2)

4. Selecting Data From Multiple Tables
    - Natural Joins
    - Join with USING Clause
    - Inner Joins with ON Clause
    - Left, Right and Full Outer Joins
    - Self Joins
    - Cross Joins
5. Nested SELECT Statements

Relational Databases

Short Overview

IT TALENTS
Training Camp

Definition of a Database

- A **relational database** is a collection of tables and relationships between them

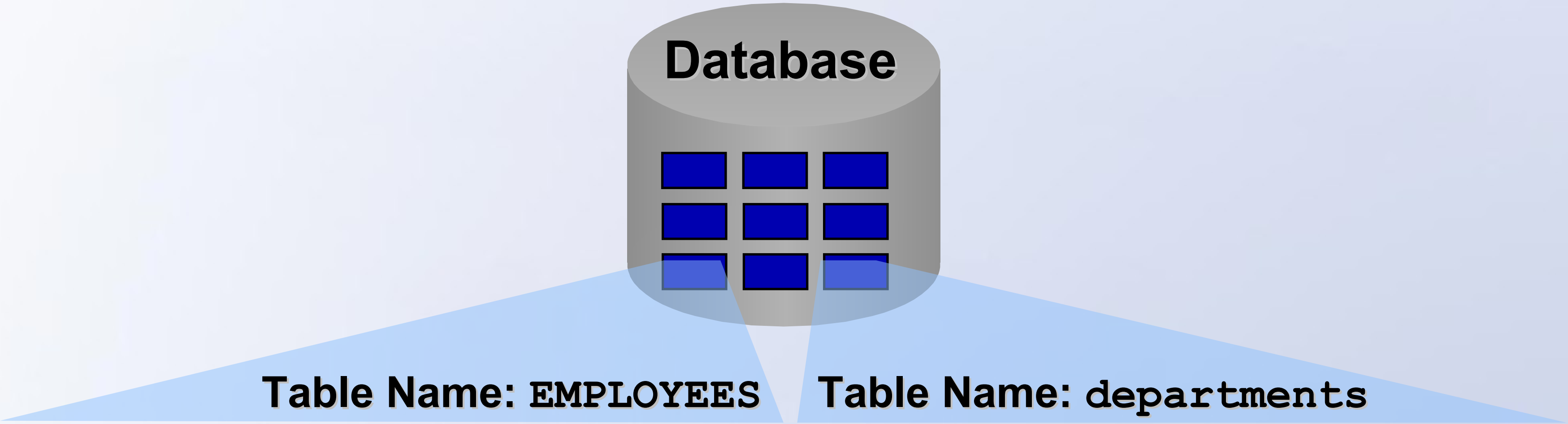


Table Name: `EMPLOYEES`

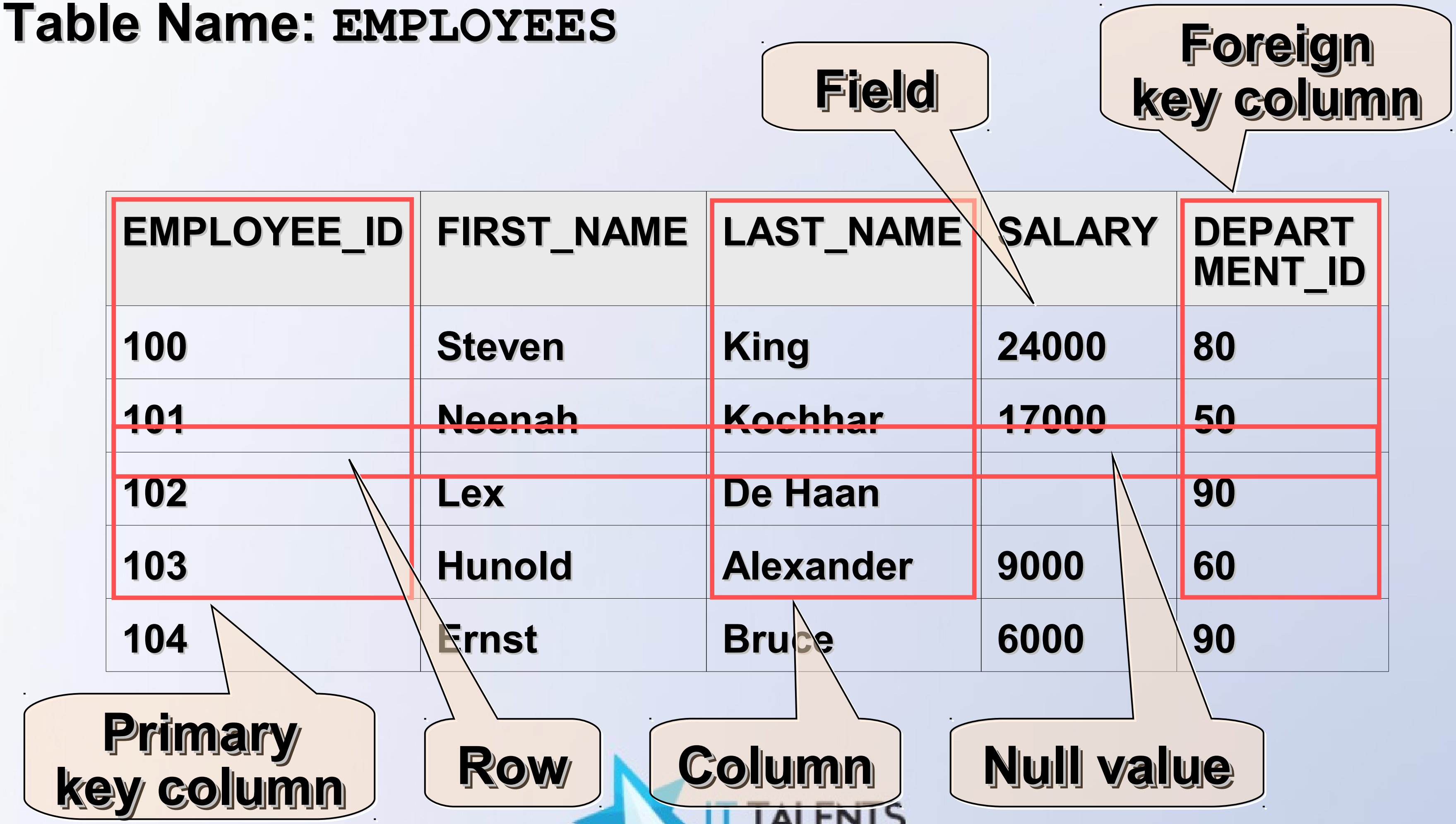
Table Name: `departments`

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
100	Steven	King	SKING
101	Neenah	Kochhar	NKOCH
102	Lex	De Haan	HAAN

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID
10	Administration	200
20	Marketing	201
50	Shipping	124

Database Tables and Terminology

Table Name: **EMPLOYEES**



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	DEPART MENT_ID
100	Steven	King	24000	80
101	Neenah	Kochhar	17000	50
102	Lex	De Haan		90
103	Hunold	Alexander	9000	60
104	Ernst	Bruce	6000	90

Field

Foreign key column

Primary key column

Row

Column

Null value

Relationships between Tables

- Each row of data in a table is uniquely identified by a primary key (PK)
- You can logically relate data from multiple tables using foreign keys (FK)

Table Name: `EMPLOYEES`

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	80
101	Neenah	Kochhar	50
102	Lex	De Haan	90

Table Name: `departments`

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
50	Shipping

Primary key

Foreign key

Primary key

MySQL Data Types

- `INT`– integer number
- `TINYINT,BIGINT` – for small or big numbers
- FLOAT, DOUBLE – floating point
- `VARCHAR2(size)` – string of variable length up to given size (locale specific)
  - `VARCHAR2(50)` – string of length up to 50
- `NVARCHAR2(size)` – Unicode string of variable length up to given size

MySQL Data Types (2)

- **`DATE` – date between Jan 1, 4712 BC and Dec 31, 9999 AD**
- **`TIMESTAMP` – date and time (year, month, day, hour, minute, and seconds)**
  - **Precision can be defined**
- **`BLOB` – binary large data object, RAW data (up to 128 TB)**
  - **Can contain photos, videos, etc.**
- **`CLOB`, `NCLOB` – character large data object (up to 128 TB)**

SQL

Introduction

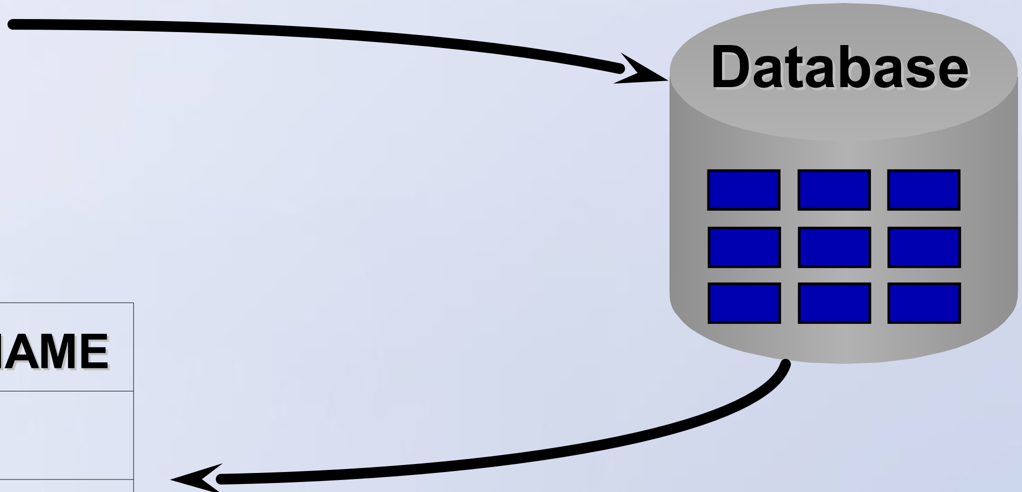
IT TALENTS
Training Camp

Communicating with a DB

SQL statement is entered

```
SELECT
DEPARTMENT_NAME
FROM departments
```

SQL statement is sent to the database



DEPARTMENT_NAME
Administration
Marketing
Shipping

The result is returned (usually as a table)

What is SQL?

- **Structured Query Language (SQL)**
  - **Declarative language for query and manipulation of relational data**
- **SQL consists of:**
  - **Data Manipulation Language (DML)**
    - **`SELECT`, `INSERT`, `UPDATE`, `DELETE`**
  - **Data Definition Language (DDL)**
    - **`CREATE`, `DROP`, `ALTER`**
    - **`GRANT`, `REVOKE`**

SQL Language

Introducing `SELECT` Statement

IT TALENTS
Training Camp

Capabilities of SQL SELECT

Projection

Take some of the columns

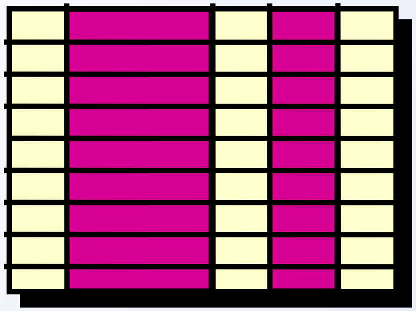


Table 1

Selection

Take some of the rows

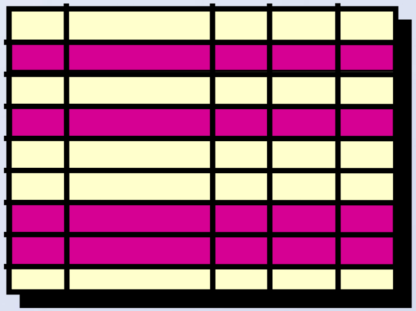


Table 1

Join

Combine tables by some column

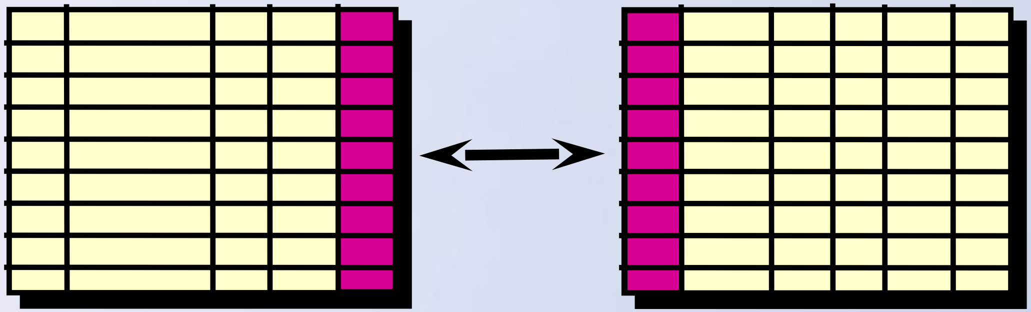


Table 1

Table 2

Basic `SELECT` Statement

```
SELECT *|{[DISTINCT] column|expression
[alias],...}
FROM table
```

- **SELECT identifies what columns**
- **FROM identifies which table**

SELECT Example

- Selecting all departments

```
SELECT * FROM departments
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1900

- Selecting specific columns

```
SELECT
   DEPARTMENT_ID,
   LOCATION_ID
FROM departments
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1900

Arithmetic Operations

- Arithmetic operators are available:
  - +, -, *, /
- Example:

```
SELECT LAST_NAME, SALARY, SALARY + 300
FROM EMPLOYEES
```

LAST_NAME	SALARY	SALARY + 300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300

The `null` Value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable
  - Not the same as zero or a blank space
- Arithmetic expressions containing a `null` value are evaluated to `null`

```
SELECT LAST_NAME, MANAGER_ID FROM EMPLOYEES
```

LAST_NAME	MANAGER_ID
King	(null)
Kochhar	100
De Haan	100

NULL is displayed as empty space or as (null)

Column Alias

- Renames a column heading
- Useful with calculations
- Immediately follows the column name
  - There is an optional AS keyword
- Double quotation marks if contains spaces

```
SELECT LAST_NAME "Name", 12*SALARY AS
  "Annual Salary" FROM employees
```

Name	Annual Salary
King	288000
Kochhar	204000

Concat

- **Concatenates columns or character strings to other columns**
- **Creates a resultant column that is a character expression**

```
SELECT CONCAT(first_name, ' ',last_name)  AS
"Employee name" FROM EMPLOYEES
```

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP

Literal Character Strings

- **A literal is a character, a number, or a date included in the `SELECT` list**
- **Date and character literal values must be enclosed within single quotation marks**
- **Each character string is output once for each row returned**

```
SELECT CONCAT(LAST_NAME, ' is a ', JOB_ID) AS
"Employee Details" FROM employees
```

Employees
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP

Removing Duplicate Rows

- The default display of queries is all rows, including duplicate rows

```
SELECT DEPARTMENT_ID
FROM employees
```

DEPARTMENT_ID
90
90
60
...

- Eliminate duplicate rows by using the `DISTINCT` keyword in the `SELECT` clause

```
SELECT
  DISTINCT DEPARTMENT_ID
FROM employees
```

DEPARTMENT_ID
90
60
...

Set Operations: UNION

- UNION combines the results from several SELECT statements
  - The columns count and types should match

```
SELECT FIRST_NAME AS NAME
FROM employees
UNION
SELECT LAST_NAME AS NAME
FROM employees
```

NAME
Abel
Adam
Alana
...

Limiting the Rows Selected

- Restrict the rows returned by using the `WHERE` clause:

```
SELECT LAST_NAME,
DEPARTMENT_ID FROM
employees WHERE
DEPARTMENT_ID = 90
```

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90

- More examples:

```
SELECT FIRST_NAME, LAST_NAME, JOB_ID FROM
employees WHERE LAST_NAME = 'Whalen'
```

```
SELECT LAST_NAME, SALARY FROM employees
WHERE SALARY <= 3000
```

Other Comparisons: BETWEEN, IN, LIKE

- Using BETWEEN operator to specify a range:

```
SELECT LAST_NAME, SALARY FROM employees
WHERE SALARY BETWEEN 2500 AND 3000
```

- Using IN / NOT IN to specify a set of values:

```
SELECT FIRST_NAME, LAST_NAME, MANAGER_ID FROM
employees WHERE MANAGER_ID IN (100, 101, 201)
```

- Using LIKE operator to specify a pattern:

```
SELECT FIRST_NAME FROM employees
WHERE FIRST_NAME LIKE 'S%'
```

  - % means 0 or more chars; _ means one char

Comparing with NULL

- Checking for NULL value:

```
SELECT LAST_NAME, MANAGER_ID FROM employees
WHERE MANAGER_ID IS NULL
```

```
SELECT LAST_NAME, MANAGER_ID FROM employees
WHERE MANAGER_ID IS NOT NULL
```

- Attention: COLUMN=NULL is always false!

```
SELECT LAST_NAME, MANAGER_ID FROM employees
WHERE MANAGER_ID=NULL
```

This is always false!

```
SELECT LAST_NAME, MANAGER_ID FROM employees
WHERE NULL=NULL
```

This is always false!

Logical operators

- Using OR and AND operators:

```
SELECT LAST_NAME, JOB_ID, SALARY FROM employees
WHERE SALARY >= 1000 AND JOB_ID LIKE '%MAN%'
```

```
SELECT LAST_NAME FROM employees
WHERE COMMISSION_PCT IS NOT NULL
  OR LAST_NAME LIKE '%S%'
```

- Using NOT operators:

```
SELECT LAST_NAME, SALARY, MANAGER_ID
FROM employees
WHERE NOT (MANAGER_ID IS NULL) AND
NOT (SALARY>10000)
```

Sorting with ORDER BY

- Sort rows with the ORDER BY clause
  - ASC: ascending order, default
  - DESC: descending order

```
SELECT LAST_NAME,
HIRE_DATE FROM employees
ORDER BY HIRE_DATE
```

LAST_NAME	HIRE_DATE
King	1987-06-17
Whalen	1987-09-17
Kochhar	1989-09-21

```
SELECT LAST_NAME,
HIRE_DATE FROM employees
ORDER BY HIRE_DATE DESC,
LAST_NAME
```

LAST_NAME	HIRE_DATE
Banda	2000-04-21
Kumar	2000-04-21
Ande	2000-03-24

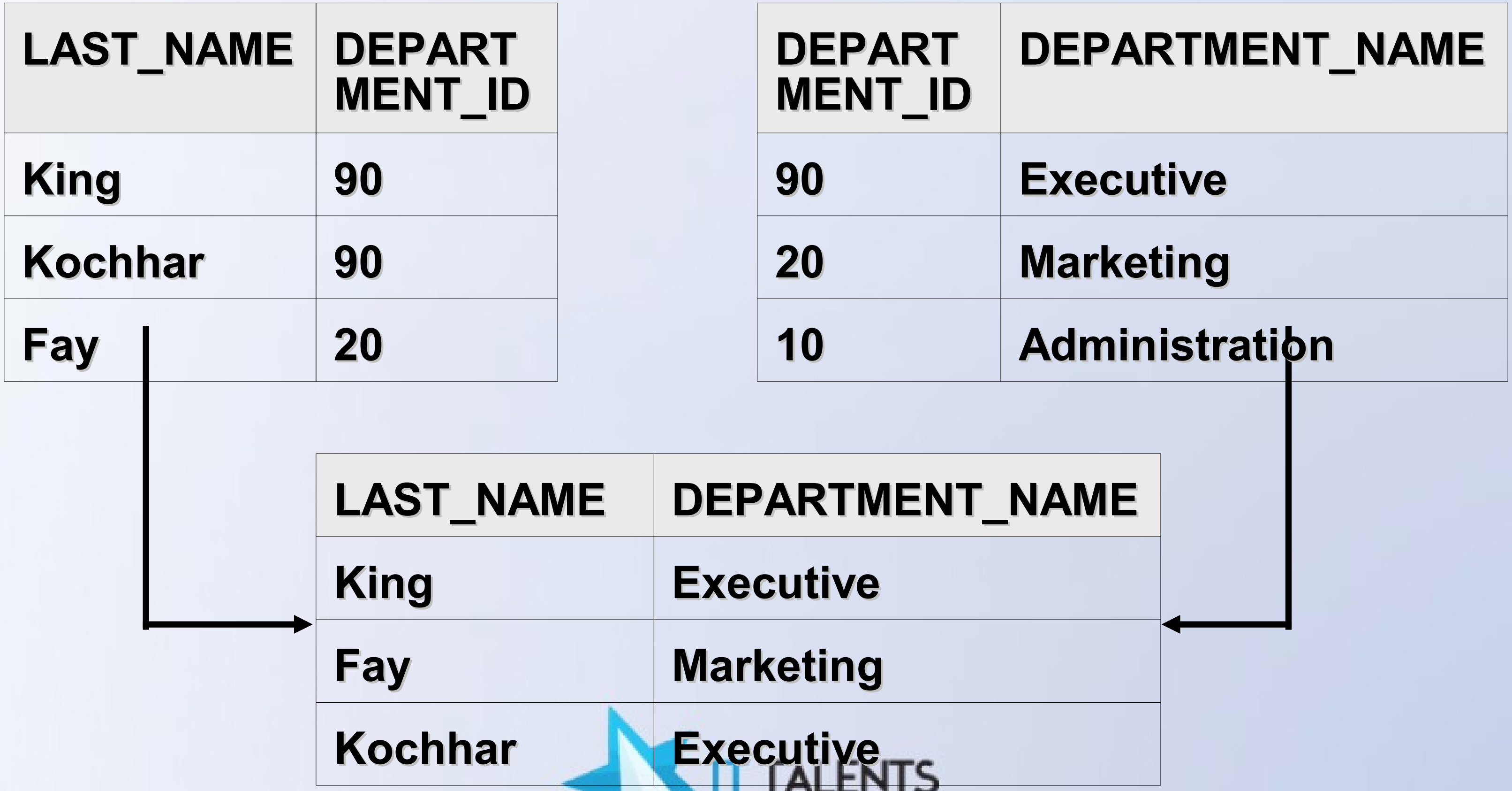
SQL Language

Selecting Data From Multiple Tables

IT TALENTS
Training Camp

Data from Multiple Tables

- Sometimes you need data from more than one table:



LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
Fay	20

DEPARTMENT_ID	DEPARTMENT_NAME
90	Executive
20	Marketing
10	Administration

LAST_NAME	DEPARTMENT_NAME
King	Executive
Fay	Marketing
Kochhar	Executive

Cartesian Product

- This will produce Cartesian product:

```
SELECT LAST_NAME, DEPARTMENT_NAME
FROM employees, departments
```

- The result:

LAST_NAME	DEPARTMENT_NAME
King	Executive
King	Marketing
King	Administration
Kochhar	Executive
Kochhar	Marketing
..	..

Cartesian Product

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition

Types of Joins

- Natural joins
- Join with `USING` clause
- Inner joins with `ON` clause
- Left, right and full outer joins
- Self joins
- Cross joins

Natural Join

- The `NATURAL JOIN` combines the rows from two tables that have equal values in all matched by name columns

```
SELECT DEPARTMENT_ID, DEPARTMENT_NAME,
  LOCATION_ID, CITY
FROM departments NATURAL JOIN LOCATIONS
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
...	...	...	...

Join with USING Clause

- If several columns have the same names we can limit the NATURAL JOIN to only one of them by the USING clause:

```
SELECT E.EMPLOYEE_ID, E.LAST_NAME,
  D.LOCATION_ID, D.DEPARTMENT_NAME
FROM employees E JOIN departments D
  USING (DEPARTMENT_ID)
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_NAME
102	De Haan	1700	Executive
103	Hunold	1400	IT
104	Ernst	1400	IT
...	...	...	...

Inner Join with ON Clause

- To specify arbitrary conditions or specify columns to join, the ON clause is used
  - Such JOIN is called also INNER JOIN

```
SELECT E.EMPLOYEE_ID, E.LAST_NAME,
  E.DEPARTMENT_ID, D.DEPARTMENT_ID, D.LOCATION_ID
FROM employees E JOIN departments D
  ON (E.DEPARTMENT_ID = D.DEPARTMENT_ID)
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800

INNER vs. OUTER Joins

- The join of two tables returning only matched rows is an inner join
- A join between two tables that returns the results of the inner join as well as unmatched rows from the left (or right) table is a left (or right) outer join
- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a *full outer join*

IT TALENTS
Training Camp

INNER JOIN

```
SELECT CONCAT(E.FIRST_NAME, ' ', E.LAST_NAME) AS
  MANAGER_NAME, D.DEPARTMENT_ID, D.DEPARTMENT_NAME
FROM employees E INNER JOIN departments D
  ON E.EMPLOYEE_ID=D.MANAGER_ID
```

MANAGER_NAME	DEPARTMENT_ID	DEPARTMENT_ NAME
Jennifer Whalen	10	Administration
Michael Hartstein	20	Marketing
Den Raphaely	30	Purchasing
Susan Mavris	40	Human Resources
Adam Fripp	50	Shipping
Alexander Hunold	60	IT
Hermann Baer	70	Public Relations
...	...	...

LEFT OUTER JOIN

```
SELECT CONCAT(E.FIRST_NAME,' ',E.LAST_NAME) AS
  MANAGER_NAME, D.DEPARTMENT_ID, D.DEPARTMENT_NAME
FROM employees E LEFT OUTER JOIN departments D
  ON E.EMPLOYEE_ID=D.MANAGER_ID
```

MANAGER_NAME	DEPARTMENT_ID	DEPARTMENT_ NAME
Jennifer Whalen	10	Administration
Michael Hartstein	20	Marketing
Den Raphaely	30	Purchasing
Clara Vishney	(null)	(null)
Jason Mallin	(null)	(null)
Hazel Philtanker	(null)	(null)
Nanette Cambrault	(null)	(null)
...	...	...

RIGHT OUTER JOIN

```
SELECT CONCAT(E.FIRST_NAME,' ',E.LAST_NAME) AS
  MANAGER_NAME, D.DEPARTMENT_ID, D.DEPARTMENT_NAME
FROM employees E RIGHT OUTER JOIN departments D
  ON E.EMPLOYEE_ID=D.MANAGER_ID
```

MANAGER_NAME	DEPARTMENT_ID	DEPARTMENT_ NAME
Jennifer Whalen	10	Administration
Michael Hartstein	20	Marketing
Den Raphaely	30	Purchasing
(null)	120	Treasury
(null)	130	Corporate Tax
(null)	140	Control And Credit
(null)	150	Shareholder Services
...	...	...

FULL OUTER JOIN

```
SELECT E.FIRST_NAME || ' ' || E.LAST_NAME AS
  MANAGER_NAME, D.DEPARTMENT_ID, D.DEPARTMENT_NAME
FROM employees E FULL OUTER JOIN departments D
  ON E.EMPLOYEE_ID=D.MANAGER_ID
```

MANAGER_NAME	DEPARTMENT_ID	DEPARTMENT_ NAME
Jennifer Whalen	10	Administration
Michael Hartstein	20	Marketing
...	...	...
Clara Vishney	(null)	(null)
Jason Mallin	(null)	(null)
...	...	...
(null)	150	Shareholder Services
...	...	...

Three-Way Joins

- A three-way join is a join of three tables

```
SELECT E.EMPLOYEE_ID, CITY, DEPARTMENT_NAME
FROM employees E
JOIN departments D
  ON D.DEPARTMENT_ID = E.DEPARTMENT_ID
JOIN LOCATIONS L
  ON D.LOCATION_ID = L.LOCATION_ID
```

EMPLOYEE_ID	CITY	DEPARTMENT_ NAME
103	Southlake	IT
104	Southlake	IT
124	San Francisco	Administration
...	...	...

Self Join

- Self join means to join a table to itself
  - Always used with table aliases

```
SELECT CONCAT(E.FIRST_NAME,' ',E.LAST_NAME,
 ' is managed by ', M.LAST_NAME) as MSG
FROM employees E JOIN employees M
ON (E.MANAGER_ID = M.EMPLOYEE_ID)
```

MSG
Neena Kochhar is managed by King
Lex De Haan is managed by King
Alexander Hunold is managed by De Haan
Bruce Ernst is managed by Hunold
..

Cross Join

- The CROSS JOIN clause produces the cross-product of two tables
  - Same as a Cartesian product
  - Not often used

```
SELECT LAST_NAME, DEPARTMENT_NAME
FROM employees CROSS JOIN departments
```

LAST_NAME	DEPARTMENT_NAME
King	Executive
King	Marketing
King	Administration
Kochhar	Executive
..	..

Additional Conditions

- You can apply additional conditions in the **WHERE** clause:

```
SELECT E.EMPLOYEE_ID,
  E.FIRST_NAME || ' ' || E.LAST_NAME AS NAME,
  E.MANAGER_ID, E.DEPARTMENT_ID, D.DEPARTMENT_NAME
FROM employees E JOIN departments D ON
   (E.DEPARTMENT_ID = D.DEPARTMENT_ID)
WHERE E.MANAGER_ID = 149
```

EMPLOYEE_ID	NAME	MANAGER_ID	DEPARTMENT_ID	DEPARTMENT_NAME
174	Ellen Abel	149	80	Sales
175	Alyssa Hutton	149	80	Sales
...	...	...	...	...

Complex Join Conditions

- Joins can apply any Boolean expression in the `ON` clause:

```
SELECT E.FIRST_NAME, E.LAST_NAME,D.DEPARTMENT_NAME
FROM employees E
  INNER JOIN departments D
  ON (E.DEPARTMENT_ID = D.DEPARTMENT_ID
  AND E.HIRE_DATE > CAST('1991-1-1' AS DATE)
  AND D.DEPARTMENT_NAME in ('Sales', 'Finance'))
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_NAME
148	Gerald	Cambrault	Sales
149	Eleni	Zlotkey	Sales
113	Luis	Popp	Finance
...	...	...	...

SQL Language

Nested `SELECT` Statements

IT TALENTS
Training Camp

Nested SELECT Statements

- SELECT statements can be nested in the where clause

```
SELECT FIRST_NAME, LAST_NAME, SALARY
FROM employees
WHERE SALARY =
   (SELECT MAX(SALARY) FROM employees)

SELECT FIRST_NAME, LAST_NAME, SALARY
FROM employees
WHERE DEPARTMENT_ID IN
   (SELECT DEPARTMENT_ID FROM departments
    WHERE DEPARTMENT_NAME='Accounting')
```

- Note: Always prefer joins to nested SELECT statements (better performance)

Using the EXISTS operator

- Using the EXISTS operator in SELECT statements
  - Find all employees that have worked in the past in the department #110

```
SELECT FIRST_NAME, LAST_NAME
FROM employees E
WHERE EXISTS
   (SELECT EMPLOYEE_ID FROM JOB_HISTORY JH
    WHERE DEPARTMENT_ID = 110 AND
    JH.EMPLOYEE_ID=E.EMPLOYEE_ID)
```

Selecting Top X Records from a Result Set

- MySQL does support `SELECT TOP X`
  - And we can use the `LIMIT` :

```
SELECT LAST_NAME, SALARY
   FROM employees
   ORDER BY SALARY DESC
   LIMIT 5
```

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Russell	14000
Partners	13500