ASIAN INSTITUTE OF MANAGEMENT

Data Mining and Wrangling

Working with databases II

Session 9

BSDSBA 2028

18 February 2026

ASIAN INSTITUTE OF MANAGEMENT

Class Administrative Matters

Course Deliverables

Assessed – R04, ICA03, E2, E1S2

Due this Week – E2S2 (Fri EOD), R05 (Wed EOD)

Upcoming – ICA04 (due: Wed next week EOD), ICA05 (due: Wed next week EOD), E3 (due Fri next week EOD),

In the near future – A1, Mini Project 1, Midterm Exam

Supplementary Materials

Git branching tutorial

Sample midterms

Observed Trends from Recent Deliverables

ICA03– Regular Expressions

Trimming whitespace – `r'^\s*(.*?)\s*$'`

E2 – Regular Expressions

`is_date` – ensure consistent date separator

`repeat_alternate` – use RegEx referencing instead of for-loop & conditional

`get_big` – use case-insensitive matching

`get_client` – too specific RegEx pattern

Session 9 and 10 – Working with databases II

Gameplan

Session 9

SQL Joins (Review)

Data Aggregations

Session 10

Inserting Data; Updating and Deleting Tables

Creating and Manipulating Tables

SQL Joins

SQL Joins

Why do we need to join tables?

Relational databases are designed such that information is split into multiple tables, one for each *entity*. Doing so enables more *efficient storage*, *easier manipulation*, and *greater stability*.

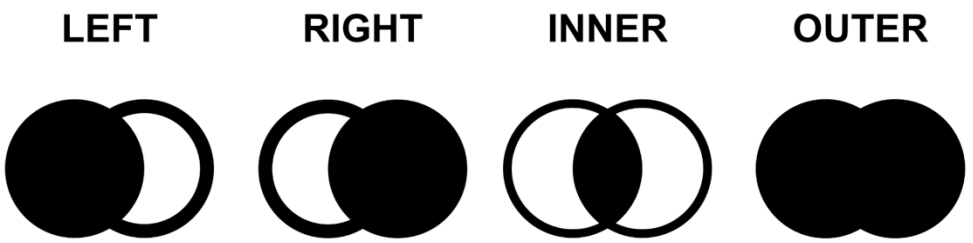
However, the price of having relational tables, is that we need a mechanism to associate or **join** tables if we want to retrieve information from multiple tables simultaneously.

SQL Joins

How do we join SQL tables?

When joining SQL tables, we need to specify three things:

1. The tables which we want to combine
2. The join relationship which acts as a filter to include only rows that matches the specified condition.
3. The type of join to use when combining the tables. This can be:



AIM

SQL Joins

Worked Example

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

To join: **journeys_end**
demon_king

Condition: **je.int = dk.int**

Join Type: **inner**

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
INNER JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int	
Frieren	3	2	5	✅
Himmel	4	4	2	❌
Heiter	2	2	4	❌
Eisen	5	4	3	❌

Condition: je.int = dk.int

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
INNER JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int	
Frieren	3	2	5	❌
Himmel	4	4	2	❌
Heiter	2	2	4	✅
Eisen	5	4	3	❌

Condition: je.int = dk.int

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
INNER JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

Condition: je.int = dk.int

SQL Joins

```
SELECT *
FROM journeys_end AS je
INNER JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

name	str	dex	int	name	str	dex	int
Frieren	3	2	5	Frieren	3	2	5
Fern	2	2	4	Heiter	2	2	4

Condition: je.int = dk.int

Join Type: inner

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
LEFT JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

name	str	dex	int	name	str	dex	int
Frieren	3	2	5	Frieren	3	2	5
Fern	2	2	4	Heiter	2	2	4
Stark	4	4	1	NULL	NULL	NULL	NULL

Condition: je.int = dk.int

Join Type: left

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
RIGHT JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

Condition: je.int = dk.int

Join Type: right

SQL Joins

```
SELECT *
FROM journeys_end AS je
RIGHT JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

Condition: je.int = dk.int

Join Type: right

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
RIGHT JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

Condition: je.int = dk.int

Join Type: right

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
RIGHT JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

Condition: je.int = dk.int

Join Type: right

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
RIGHT JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

Condition: je.int = dk.int

Join Type: right

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
RIGHT JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

name	str	dex	int	name	str	dex	int
Frieren	3	2	5	Frieren	3	2	5
NULL	NULL	NULL	NULL	Himmel	4	4	2
Fern	2	2	4	Heiter	2	2	4
NULL	NULL	NULL	NULL	Eisen	5	4	3

Condition: je.int = dk.int

Join Type: right

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
FULL OUTER JOIN demon_king AS dk
    ON je.int = dk.int
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

name	str	dex	int	name	str	dex	int
Frieren	3	2	5	Frieren	3	2	5
NULL	NULL	NULL	NULL	Himmel	4	4	2
Fern	2	2	4	Heiter	2	2	4
NULL	NULL	NULL	NULL	Eisen	5	4	3
Stark	4	4	1	NULL	NULL	NULL	NULL

Condition: je.int = dk.int

Join Type: full outer

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
LEFT JOIN demon_king AS dk
    ON je.dex = dk.dex
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

name	str	dex	int	name	str	dex	int

Condition: je.dex = dk.dex

Join Type: left

AIM

SQL Joins

```
SELECT *
FROM journeys_end AS je
LEFT JOIN demon_king AS dk
    ON je.dex = dk.dex
```

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

name	str	dex	int	name	str	dex	int
Frieren	3	2	5	Frieren	3	2	5
Frieren	3	2	5	Heiter	2	2	4
Fern	2	2	4	Frieren	3	2	5
Fern	2	2	4	Heiter	2	2	4
Stark	4	4	1	Himmel	4	4	2
Stark	4	4	1	Eisen	5	4	3

Condition: je.dex = dk.dex

Join Type: left

AIM

SQL Joins

Exercise

What is the result of the following SQL join operation?

journeys_end

name	str	dex	int
Frieren	3	2	5
Fern	2	2	4
Stark	4	4	1

demon_king

name	str	dex	int
Frieren	3	2	5
Himmel	4	4	2
Heiter	2	2	4
Eisen	5	4	3

To join: **journeys_end** **demon_king**

Condition: **je.int ≤ dk.int**

Join Type: **left**

AIM

Data Aggregations

Data Aggregations

What are aggregate functions?

Aggregate functions are functions that take many rows of data then return one summarized value. They are used mainly to summarized data and describe it without having to show its contents.

Data Aggregations

What are some examples of useful data aggregations?

- Determining the number of rows in a table (or the number of rows that meet some condition).
- Obtaining the sum of a set of rows in a table.
- Finding the highest, lowest, and average values in a table column (either for all rows or for specific rows).

Data Aggregations

SQL Aggregate Functions

Function	Description
`AVG()`	Returns the average value
`COUNT()`	Returns the number of rows
`MAX()`	Returns the highest value
`MIN()`	Returns the lowest value
`SUM()`	Returns the sum of values

Data Aggregations

Why use GROUP BY?

`GROUP BY` is useful when you want to aggregate at a specific level of your data. `GROUP BY` allows you to divide your data into logical sets so you can perform aggregate calculations on each group.

Data Aggregations

Worked Example

characters

name	class	str	dex	int
Frieren	Mage	3	2	5
Himmel	Paladin	4	4	2
Eisen	Barbarian	5	4	3
Fern	Mage	2	2	4
Stark	Barbarian	4	4	2
Flamme	Mage	3	3	5

Data summarization

```
SELECT
    AVG(str) AS avg_str,
    AVG(dex) AS avg_dex,
    AVG(int) AS avg_int
FROM characters
```

avg_str	avg_dex	avg_int
3.5	3.166667	3.5

AIM

Data Aggregations

Worked Example

characters

name	class	str	dex	int
Frieren	Mage	3	2	5
Himmel	Paladin	4	4	2
Eisen	Barbarian	5	4	3
Fern	Mage	2	2	4
Stark	Barbarian	4	4	2
Flamme	Mage	3	3	5

Data summarization at `class` level

```
SELECT
    class,
    AVG(str) AS avg_str,
    AVG(dex) AS avg_dex,
    AVG(int) AS avg_int
FROM characters
GROUP BY class
```

AIM

characters

name	class	str	dex	int
Frieren	Mage	3	2	5
Himmel	Paladin	4	4	2
Eisen	Barbarian	5	4	3
Fern	Mage	2	2	4
Stark	Barbarian	4	4	2
Flamme	Mage	3	3	5

Data summarization at `class` level

```
SELECT
    class,
    AVG(str) AS avg_str,
    AVG(dex) AS avg_dex,
    AVG(int) AS avg_int
FROM characters
GROUP BY class
```

GROUP BY class

name	class	str	dex	int
Frieren	Mage	3	2	5
Fern	Mage	2	2	4
Flamme	Mage	3	3	5

name	class	str	dex	int
Eisen	Barbarian	5	4	3
Stark	Barbarian	4	4	2

name	class	str	dex	int
Himmel	Paladin	4	4	2

Apply aggregation functions

class	avg_str	avg_dex	avg_int
Mage	2.66667	2.333333	4.666667

class	avg_str	avg_dex	avg_int
Barbarian	4.5	4	2.5

class	avg_str	avg_dex	avg_int
Paladin	4	4	2

Combine into final output

class	avg_str	avg_dex	avg_int
Mage	2.66667	2.333333	4.666667
Barbarian	4.5	4	2.5
Paladin	4	4	2

ASIAN INSTITUTE OF MANAGEMENT

Data Mining and Wrangling

Working with databases II

Session 10

BSDSBA 2028

18 February 2026

ASIAN INSTITUTE OF MANAGEMENT

Data Aggregations

Rules to consider when using `GROUP BY`

- `GROUP BY` clauses can contain as many columns as you want.
- If the grouped column contains a row with a `NULL` value, `NULL` will be returned as a group.
- The `GROUP BY` clause must come after any `WHERE` clause and before any `ORDER BY` clause.

Inserting Data

Inserting Data

How do we insert data into an SQL table?

We use the `INSERT INTO` statement which can be used in several ways:

- Inserting a single complete row
- Inserting a single partial row
- Inserting the results of a query

Updating and Deleting Data

Updating and Deleting Data

Rules to live by

- Never execute an `UPDATE` or `DELETE` without a `WHERE` clause.
- Make sure every table has a primary key and use it as filter in the `WHERE` clause.
- Before using `UPDATE` or `DELETE`, check the coverage of the `WHERE` filter using a `SELECT` statement.
- Use database-enforced referential integrity so that your DBMS will not allow deletion of rows that have data in other tables related to them.

Creating and Manipulating Tables

Creating and Manipulating Tables

Rules to live by

- Ideally, tables should *never* be altered after they contain data.
- Always use primary keys and enforce relationships with foreign keys.
- Be explicit with `NULL` versus `NOT NULL`.
- Be mindful in choosing the right data types.
- Name things clearly and consistently and add documentation as needed.
- Back up before doing big changes.

Reminders

Next Session

Session 11 and 12 (March 4, 2026) – Web scraping: HTML, bs4, requests

Deliverables

R05 – Working with Databases II (due: Wednesday, February 18, 2026, EOD)

E2S2 – Regular Expressions (due: Friday, February 20, 2026, EOD)

ICA04 – Working with Databases I (due: Wednesday, February 25, 2026, EOD)

ICA05 – Working with Databases II (due: Wednesday, February 25, 2026, EOD)

E3 – Working with Databases (due: Friday, February 27, 2026, EOD)