THE UNIVERSITY OF
WAIKATO
*Te Whare Wānanga o Waikato*

DEPARTMENT OF
COMPUTER SCIENCE
*Te Tari Rorohiko*

**2025**

Contributors

J. Turner

R. Mercado

V. Moxham-Bettridge

A. Hinze

C. Pilbrow

N. Kanji

T. Elphick

S. Cunningham

J. Kasmara

# INTRODUCTION TO DATABASES

Last week we looked into human-robot interaction with the robot, Cruz. Today we look at an introduction to databases, how we can design them and write well-formatted SQL queries.

## What is a database?

Last week we looked into testing and safety-critical interactive systems. Today we look at an introduction to databases, how we can design them and write well-formatted SQL queries. According to Oracle (n.d.), “A database is an organised collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications associated with them, are referred to as a database system, often shortened to just database”[1].

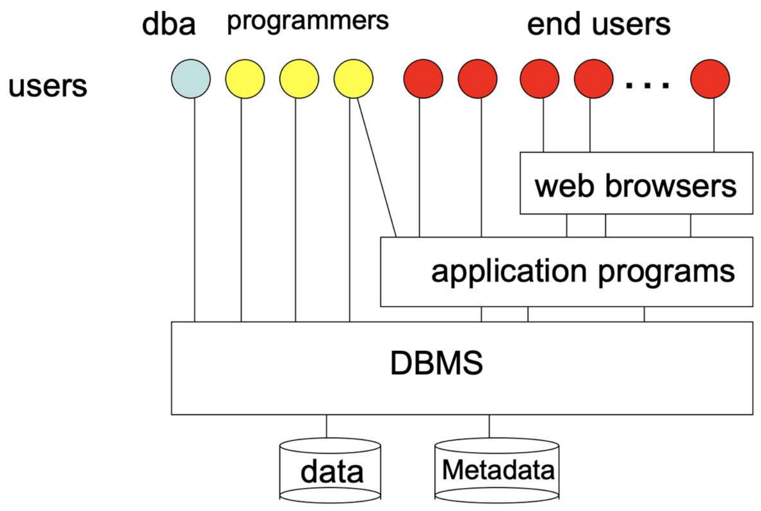


*Figure 45: A Standard Database Application*

Figure 45 demonstrates how a standard database application might look like. Typically, users interact with a web application inside of a web browser and that application interacts with a DBMS to get information from the database to display to a user. This may also include gathering information from the user and then storing that information in a database. Think of websites you might commonly interact with in NZ, like TradeMe or Air New Zealand. These web applications are connected to databases to store listing and flight information respectively. Users may wish to

[1] https://www.oracle.com/za/database/what-is-database/

view that information or add a new listing or booking, it is the database's job to store that information correctly.

## How do we create a database?

Databases come in many different forms depending on your design choice, however, one of the most common types is a relational database. This is where we exploit relationships between different entities, an entity being something we want to store information about. For example:

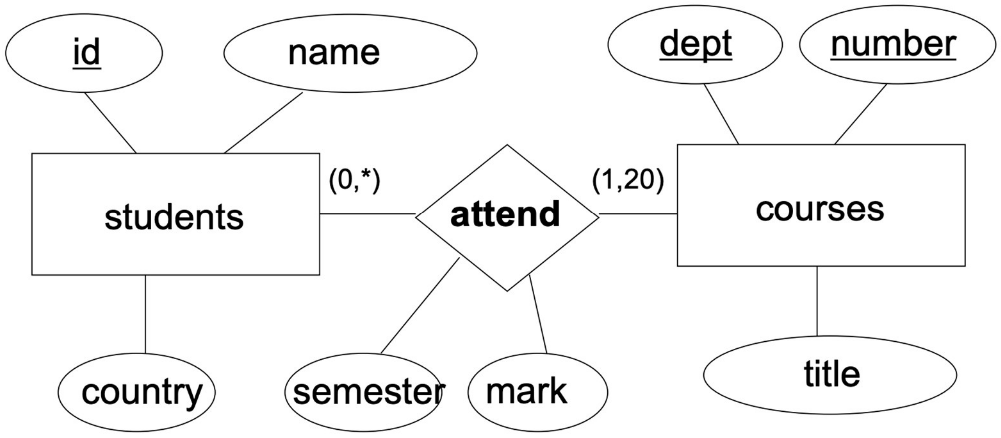


*Figure 46: Entity-Relationship Diagram (ERD) Example*

In Figure 46 we have a very simple example of a database which stores information about students and the courses that they attend. A rectangular box represents an entity, something we want to store information about. A diamond box represents a relationship between two entities, in this case the fact that students attend courses.

Both entities and relationships may have attributes (the pieces of information we want to store) and these are represented by ovals. For students we keep track of their ID, name and country. For courses, we keep track of the department, number and title. For the attend relationship we keep track of the semester and mark.

You will see that on either side of the attend relationship we have cardinalities specified (numbers inside of the brackets). A cardinality specifies how many of that entity the other sees. In this example students attend at most 1 to 20 courses, while courses may have 0 to many students enrolled.

You will notice that some of the attributes are underlined, these represent what we refer to as a primary key. A primary key is a unique identifier for a particular record stored in a database. We only require primary keys for entities and not relationships in an ERD. This will become more clear as we translate the ERD to a relational model which allows us to define the database.

There are 6 steps we must follow in order to create an ERD:

1. Start with a description of the database.
2. Identify entities (they are often the important nouns).
3. Identify relationships (they are often verbs).
4. Identify attributes (attributes describe entities/relationships).
5. Identify keys (something unique that can be used to identify the entity).
6. Find cardinalities (how many of an entity exists in regards to a specific relationship).

**Exercises:**

1. Let's consider the following new information about our student system from Figure 32 and determine what the new ERD would look like:

   "Students take a course and receive a grade for their performance. A student may only take up to 20 courses, however there is no limit to how many students may attend a course. Students have a unique student ID, a name and country. Courses have a unique department, number and a title. Students may enrol in the same course over different semesters. Each course is supervised by up to 3 lecturers. Each lecturer has a unique staff ID, a name and department. Both students and lecturers have a contact email."

## From ERD to Relational Model

The ERD is an important tool used to clarify the design of a relational database, however, in order to define tables we need to understand how these relationships organise data and the logical links between them.

A relational model defines three different aspects of the database design: **relations,** data organised in tables; **attributes,** every table with a fixed set of columns; and **Foreign Keys,** logical links between tables.

Let's consider the 5 steps required to take an ERD to a Relational model:

1. For each entity A define a relation (table) such that: *A(attributes of A)*.
2. For each many to many relationship between entities A and B, define a relation (table) *R1(key-attribute of A, key-attribute of B, attributes of relationship,...)* where the key is formed by the keys of both entities.

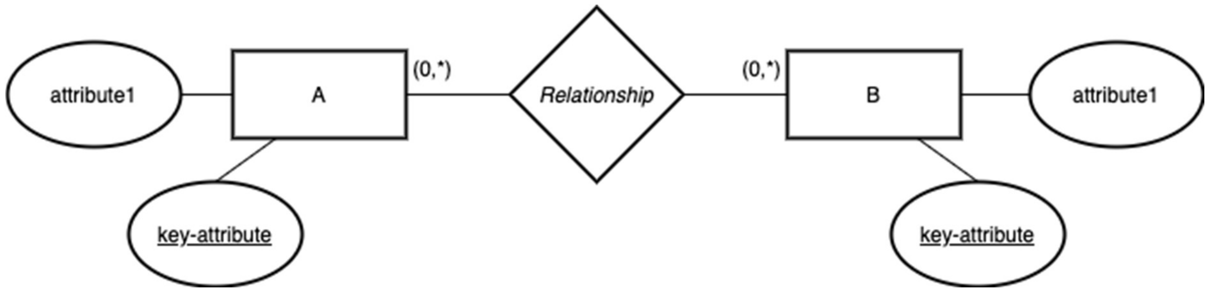


*Figure 47: Many-to-Many Relationship*

3. For each one to many relationship R2 between entities A and B, the key of A becomes an attribute of B such that *B(attributes of B, key-attribute of A)*

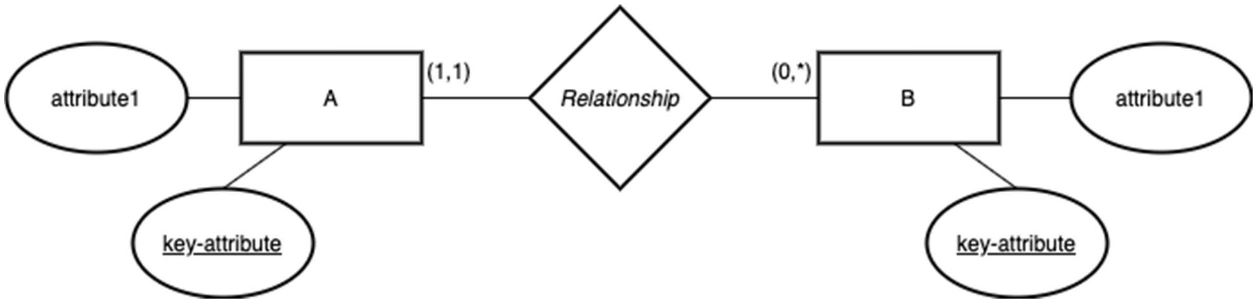


*Figure 48: One-to-many Relationship*

4. For each one to one relationship R3 between entities A and B, either the key of A becomes an attribute of B, or the key of B becomes an attribute of A such that *B(attributes of B, key-attribute of A)* OR *A(attributes of A, key-attribute of B)*

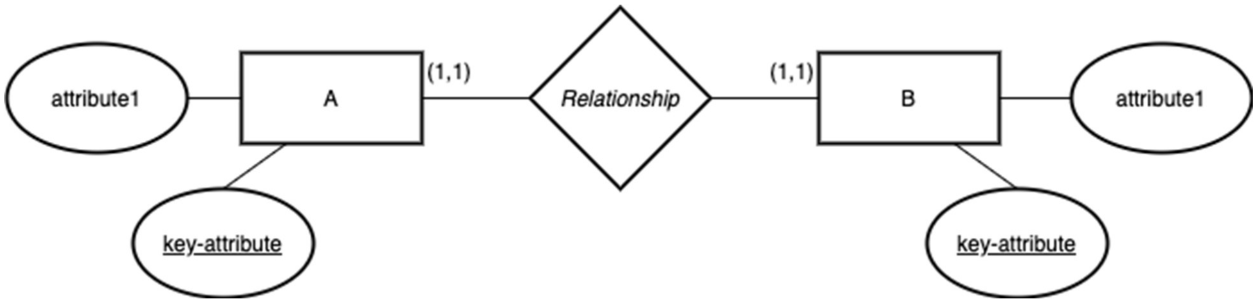


*Figure 49: One-to-one Relationship*

5. Normalisation, we will only focus on a simple normalisation technique, if we have two tables with the same keys they can be combined.

Using the above rules we consider the example from figure 46 and what this would look like as a relational model, can you work out how the rules were applied?

*students(<u>id</u>, name, country)*

*courses(<u>dept</u>, <u>number</u>, title)*
*attend(semester, mark, <u>student-id</u>, <u>courses-dept</u>, <u>courses-number</u>)*

For our attend relationship we need to determine which of the steps to apply. The attend relationship is many to many because a course has the potential for many students while a student may enrol in up to 20 courses, therefore, we need the keys from both courses and attend in our table. In this instance none of the keys can be normalised (as none of the keys are identical).

Note that this design is ideal, we can search all students, all courses, and the relationship table of attend allows us to determine which students have been enrolled in which courses. It does this by associating the primary keys of our two entities.

**Exercises:**

1. Consider the more complex ERD for a swimming lessons system shown in Figure 36. Convert the diagram to a relational model.

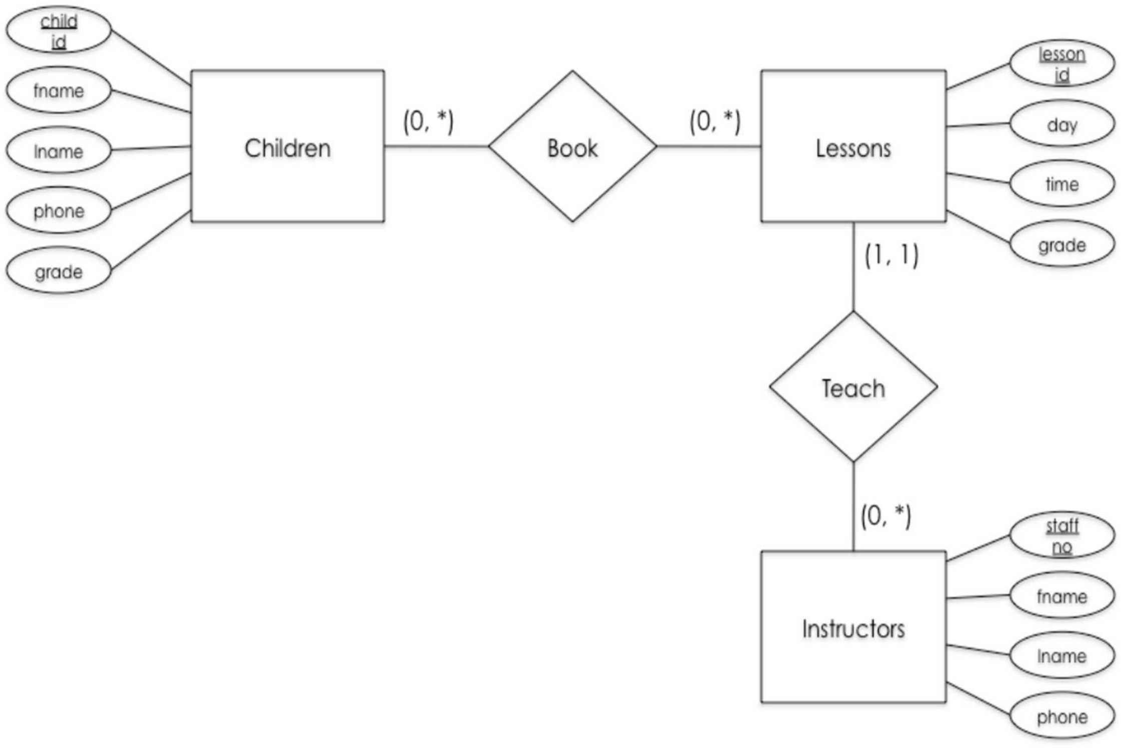
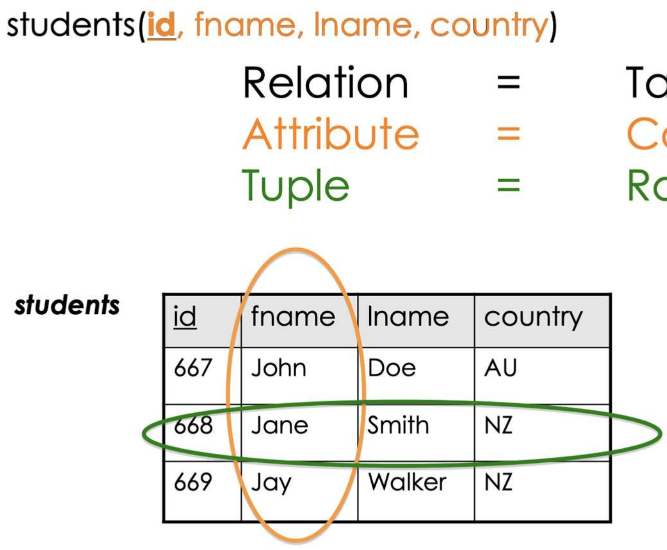


*Figure 50: Swimming Lessons ERD*

## Relational Model to SQL Database

Thus far we have mentioned SQL but not explained what this actually means. SQL stands for **S**tructured **Q**uery **L**anguage and is pronounced "S-Q-L" or "sequel". It is the standard for relational DB systems and built on a declarative programming paradigm, meaning that we declare what we want from the computer but do not define how to actually go about this (in contrast to the C# examples we have seen in earlier sessions).

For each relation in our relational model we define a table in SQL where the relation name becomes the name of the table; an attribute becomes a column of the table; and a tuple becomes a row (a tuple can be thought of as an individual record of data). For example:



| id | fname | lname | country |
|---|---|---|---|
| 667 | John | Doe | AU |
| 668 | Jane | Smith | NZ |
| 669 | Jay | Walker | NZ |

*Figure 51: Relation to Table*

In Figure 51 we can see that the table students has columns id, fname, name and country. A row of the table "668, Jane, Smith, NZ" defines the data for a single student. Let's use our running example of the student system and create a database in Oracle using livesql.oracle.com, you will need to create an oracle account to get started.

**Exercises:**

1. Consider our relational model students "*students(id, name, country)*" we will use the create table command to generate a table, enter the following code into your SQL worksheet:

```
CREATE TABLE students (
     student_id NUMBER GENERATED BY DEFAULT AS IDENTITY,
     full_name VARCHAR2(100) NOT NULL,
     country VARCHAR2(50) NOT NULL,
     PRIMARY KEY(student_id)
);
```

2. Now we need to create the tables for the courses and attend relations:

```
CREATE TABLE courses (
     depart VARCHAR2(100) NOT NULL,
     courseNum NUMBER(3) GENERATED BY DEFAULT AS IDENTITY,
     title VARCHAR2(1000) NOT NULL,
     PRIMARY KEY(depart,courseNum)
);

CREATE TABLE attend (
    semester VARCHAR(1) NOT NULL,
    mark NUMBER(3),
    student_id NUMBER,
    depart VARCHAR2(100) NOT NULL,
    courseNum NUMBER(3),
    PRIMARY KEY(student_id,depart,courseNum),
    CONSTRAINT fk_student_id FOREIGN KEY (student_id)
REFERENCES students(student_id),
    CONSTRAINT fk_course FOREIGN KEY (depart,courseNum)
REFERENCES courses(depart,courseNum)
);
```

Note, if you are having errors occur because a table already exists you can remove that table by using the command:

```
DROP TABLE students CASCADE CONSTRAINTS;
```

3. Download a copy of the `studentsystem.sql` script from the Slack channel for this session, copy and paste the script and then hit run. Now we have rows of data in our student system we can start querying our database.

## How do we query databases?

Now that we have information stored in the database we can start making queries to retrieve information. While queries are made up of different statements (or commands) the simplest form is a “SELECT, FROM” query. Let’s look at an example:

```
SELECT * FROM students;
```

| STUDENT_ID | FULL_NAME | COUNTRY |
|---|---|---|
| 1 | Hunter Klein | Seychelles |
| 2 | Drew Shannon | Marshall Islands |
| 3 | Gregory Horn | Turkmenistan |
| ... | ... | ... |

*Figure 52: Select From Query Results*

This query says get all of the rows of information (tuples) in the database from the table students. Figure 52 gives an example of what the results may look like as a table. We can also add a "WHERE" clause which specifies retrieving a specific piece of information based on a column. For example:

```
SELECT * FROM students WHERE student_id <= 10;
```

In this query we are still selecting all rows from the students table, however we are only asking for student's whose ID number is less than or equal to 10. More generally a query format can be considered as:

```
SELECT <column_name>
FROM <table_name>
WHERE <search_condition>
GROUP BY <column_name>
HAVING <search_condition>
ORDER BY <column_name>
```

The result from a query is always a table. Note that we can also join tables, meaning that we can find out the details of students enrolled in courses. For example:

```
SELECT *
FROM students, attend
WHERE students.student_id = attend.student_id;
```

| STUDENT_ID | FULL_NAME | COUNTRY | SEMESTER | MARK | STUDENT_ID | DEPART | COURSE NUM |
|---|---|---|---|---|---|---|---|
| 5 | Ira Dunlap | Bouvet Island | H | 24 | 5 | Data Analytics | 79 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | Sawyer Ellis | Aruba | G | 58 | 8 | Computer Science | 12 |
| 16 | George Leon | Cambodia | A | 29 | 16 | Computer Science | 67 |
| ... | ... | ... | ... | ... | ... | ... | ... |

*Figure 53: Join Query Results*

This will return all the students who are actually attending courses as opposed to our original query (`SELECT * FROM students;`) which returned all students in the system, including those that had not yet attended a course. If we just wanted the names of the students who were attending courses we would need to change our query to use the DISTINCT keyword, so that students enrolled in more than one course would not appear twice. For example:

```
SELECT DISTINCT students.full_name
FROM students, attend
WHERE students.student_id = attend.student_id;
```

| FULL_NAME |
|---|
| Sawyer Ellis |
| Philip Mcbride |
| Brenna Atkins |
| ... |

*Figure 54: Distinct Query Results*

**Exercises:**

Can you write SQL queries for the following? Try them out in the livesql window to see if you get the correct results.

1. Display all departments offering courses.
2. Display all the semesters being attended.
3. Display all students and order by their full_name (e.g. `ORDER BY full_name ASC`).
4. Display all the students who live in the Cook Islands or Marshall Islands (*hint: use the OR keyword in your WHERE clause).*
5. Display all students whose ID is greater than 10 but less than 30 *(hint: use the AND keyword in your WHERE clause).*
6. List the courses not from the Computer Science department *(hint: use the NOT keyword in your WHERE clause).*

7. Display the list of all courses titles that have students enrolled *(hint: you will need to join the attend and courses tables)*.
8. For each student, display their full name and id as well as the title of courses that they attend. Order the table by the student’s full name.
9. Following on from the previous query, select the students full name and count the number of courses they are enrolled in (`count(courses.title)`). Group this query by the students full name.
10. Using your query from part 9, display only the students which are enrolled in more than one course by adding a HAVING clause.

## Summary

In this session we have rather quickly covered a broad range of information about databases and the basics of SQL. We have used Oracle in this example but it is worth noting that lots of different database management systems exist (MongoDB, Microsoft SQL and so on). In addition, while we have covered ERDs and relational models for a relational database, non-relational databases exist and are used quite frequently amongst other types of databases.

Also note, that while we have covered the basics of ERDs and relational models we have not covered what makes a good database design. This is another whole topic on it’s own and we simply do not have the time to cover it here. If you are interested in learning more about design have a look at the additional resources linked below.

## Advanced Exercises

1. Can you determine the number of courses per department?
2. Can you determine the number of students per department?
3. Can you display all students' names who received a passing grade (greater than 50) in the data analytics department?
4. Can you count all the students which are not attending a course? *(Hint: you will need a nested query using the IN keyword!).*
5. Can you count all the courses which are not attended?
6. Can you find students which have attended all courses? *(Hint: you will need to use double negation on EXISTS keyword!).*

## Useful Resources

- Introduction to Oracle SQL: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/Introduction-to-Oracle-SQL.html#GUID-049B7AE8-11E1-4110-B3E4-D117907D77AC
- W3 Schools SQL Tutorials: https://www.w3schools.com/sql/default.asp

- Viescas, J. (2018). SQL Queries for mere mortals, 4th ed.. Can be purchased at the following link: https://www.pearsoned.co.nz/9780134858333
- Garcia-Molina, H., Ullman, J. D., & Wisdom, J. (2011). Database Systems: The Complete Book, 2nd ed.. Can be purchased at the following link: https://www.amazon.com/Database-Systems-Complete-Hector-Garcia-Molina-ebook-dp-B009TGBW58/dp/B009TGBW58/ref=mt_other?_encoding=UTF8&me=&qid=