Notes - Week 1

A summary of the first ten chapters in Rockoff's *The Language of SQL*.

General `SELECT` query

```
SELECT columnlist
FROM tablelist
WHERE condition
GROUP BY columnlist
HAVING condition
ORDER BY columnlist
```

1. Relational databases and SQL

Relational databases

A **relational database** is a collection of data stored in any number of *tables* that are *related* to each other via *columns* representing common features.

For example, the following two tables are related to each other via the column *CustomerID*.

Customers

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper

Orders

OrderID	CustomerID	OrderAmount
1	1	50.00
2	1	60.00
3	2	33.50
4	3	20.00

The **primary key** of a table is the column that contains the identification for each row in the table and is used to relate tables to each other. In the above example, *CustomerID* and *OrderID* is the primary key of *Customers* and *Orders* respectively. Notice how *CustomerID* is used to relate *Orders* to *Customers*. Primary keys are often automatically incremented (*auto-increment*).

Every column in a table has a **datatype**. The main categories of data types are:

- Numeric (may be used for arithmetic calculations)
  - Bits (0 or 1)
  - Integers
  - Decimals
  - Real numbers (only approximate value stored)
- Character
- Date / Time

Another attribute of each column is whether it is allowed to contain **NULL** values (instances of missing data).

SQL

SQL is a computer language for maintaining and extracting data from relational databases. SQL has three main components:

1. *Data manipulation language* (DML) for retrieving, updating adding and deleting data in the database.

2. *Data definition language* (DDL) for creating and modifying the actual database.

3. *Data control language* (DCL) for managing access permissions and security.

2. Basic data retrieval

`SELECT` and `FROM` are used to retrieve data from a database.

Example:

```sql
SELECT *
FROM Customers;
```

Returns everything from the *Customers* table:

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper

Example:

```sql
SELECT
  FirstName,
  LastName
FROM Customers;
```

Returns columns *FirstName* and *LastName* from the *Customers* table:

FirstName	LastName
William	Smith
Natalie	Lopez
Brenda	Harper

If a column name has an embedded space it must be contained in implementation-dependent special characters. For example, in MS SQL Server the column name should be contained in square brackets; in Oracle, double quotes; in MySQL, accent graves (`).

3. Calculations and aliases

Let's say that we have the following table in our database:

Orders

OrderID	FirstName	LastName	QuantityPurchased	PricePerItem
1	William	Smith	4	2.50
2	Natalie	Lopez	10	1.25
3	Brenda	Harper	5	4.00

Arithmetic calculations

Arithmetic calculations can be performed on numeric datatypes.

Example:

```
SELECT
  OrderID,
  QuantityPurchased,
  PricePerItem,
  QuantityPurchased * PricePerItem
FROM
  Orders;
```

Returns:

OrderID	QuantityPurchased	PricePerItem	(no column name)
1	4	2.50	10.00
2	10	1.25	12.50
3	5	4.00	20.00

Arithmetic calculations are done using the `+`, `-`, `*` and `/` operators.

Concatenation

Concatenation can be performed on character columns.

Example:

```
SELECT
  OrderID,
  FirstName,
  LastName,
  CONCAT (FirstName, ' ', LastName)
FROM
  Orders;
```

Returns:

OrderID	FirstName	LastName	(no column name)
1	William	Smith	William Smith
2	Natalie	Lopez	Natalie Lopez
3	Brenda	Harper	Brenda Harper

Some implementations use the `+` (e.g. MS SQL Server) or `||` (e.g. Oracle) operators instead of the function `CONCAT`.

Column aliases

Calculated columns can be given explicit names using the `AS` keyword.

Example:

```
SELECT
  OrderID,
  QuantityPurchased,
  PricePerItem,
  QuantityPurchased * PricePerItem AS 'Total'
FROM
  Orders;
```

Returns:

OrderID	QuantityPurchased	PricePerItem	Total
1	4	2.50	10.00
2	10	1.25	12.50
3	5	4.00	20.00

Aliases can also be useful to give a column a name that is more meaningful in a specific context, e.g.

```
SELECT x AS 'Quantity' FROM CrypticTable;
```

`AS` can also be used to give a table an alias, e.g.

```
SELECT x AS 'Quantity' FROM CrypticTable AS Orders;
```

Notice that, unlike column aliases, table aliases are not enclosed in quotes.

Prefixes

We can give selected columns a prefix using `.`, which can be useful when selecting data from several tables. For example:

```
SELECT
  Orders.LastName
FROM Orders
```

4. Using functions

There are two types of SQL functions:

1. *Scalar* functions act on data from a single row.
2. *Aggregate* functions act on entire columns.

Common *character* functions

Some common scalar functions that act on *character* data are:

- `LEFT(Value, Characters)`
- `RIGHT(Value, Characters)`
- `SUBSTRING(Value, Start, Length)`
- `LTRIM(Value)`
- `RTRIM(Value)`
- `CONCAT(Value1, Value2, ...)`
- `UPPER(Value)`
- `LOWER(Value)`

Let's sat that we have the following table in our database:

Customers

CustomerID	FirstName	LastName
1	William	Smith
2	Natalie	Lopez
3	Brenda	Harper

Example:

```
SELECT
  FirstName,
  LEFT(FirstName, 3) AS FirstThree
FROM Customers;
```

Returns:

CustomerID	FirstName	FirstThree
1	William	Wil
2	Natalie	Nat
3	Brenda	Bre

(Note: Oracle does not have `LEFT` and `RIGHT` defined; use `SUBSTR` instead.)

Common *date / time* functions

Date / time functions differ across implementations. In MySQL the function `NOW()` returns the current date-time in the format `YYYY-MM-DD hh:mm:dd`.

A useful MySQL date format function is `DATE_FORMAT(Date, Format)` which returns a representation of *Date* specified by *Format*. For format specifier meanings, see this MySQLTutorial page.

Common *numeric* functions

Some common numeric scalar functions are:

- `ROUND(Value, DecimalPlaces)`
- `RAND([Seed])`

Conversion

The function `CAST(Value AS DataType)` can be used to convert datatypes.

Displaying missing data

Let's say that we have the following table in our data base:

Products

Product	Description	Color
1	Chair A	Red
2	Chair B	NULL
3	Lamp C	Green

We can display `NULL` values as something different using the `IFNULL(Column, DisplayIfNull)` function. For example:

```
SELECT
  Description,
  IFNULL(Color, 'Unknown') AS Color
FROM Products;
```

Returns:

Product	Description	Color
1	Chair A	Red
2	Chair B	Unknown
3	Lamp C	Green

(Note that this function is called `ISNULL` in MS SQL Server and `NVL` in Oracle.)

5. Sorting data

`ORDER BY` can be used to order the data returned from a query.

Let's say that we have the following table in our database:

Customers

CustomerID	FirstName	LastName
1	William	Smith
2	Janet	Smith
2	Natalie	Lopez
3	Brenda	Harper

Then the query:

```
SELECT
  FirstName,
  LastName
FROM Customers
SORT BY LastName;
```

Returns:

FirstName	LastName
Brenda	Harper
Natalie	Lopez
William	Smith
Janet	Smith

We can use the keyword `ASC` (default) or `DESC` to determine whether the order should be ascending or descending. The query:

```
SELECT
  FirstName,
  LastName
FROM Customers
SORT BY LastName DESC;
```

Returns:

FirstName	LastName
William	Smith
Janet	Smith
Natalie	Lopez
Brenda	Harper

We can sort by multiple columns, e.g.

```
SELECT
  FirstName,
  LastName
FROM Customers
SORT BY LastName, FirstName;
```

Returns:

FirstName	LastName
Brenda	Harper
Natalie	Lopez
Janet	Smith
William	Smith

6. Column based logic

The `CASE` keyword can be used to apply logic to a query.

Let's say that we have the following table in our database:

ProductID	CategoryCode	ProductDescription
1	F	Apple
2	F	Orange
3	S	Mustard
4	V	Carrot

Then the query:

```
SELECT
  CASE CategoryCode
    WHEN 'F' THEN 'Fruit'
    WHEN 'V' THEN 'Vegetable'
    ELSE 'Other'
  END AS 'Category',
  ProductDescription AS 'Description'
FROM Products;
```

Returns:

Category	Description
Fruit	Apple
Fruit	Orange
Other	Mustard
Vegetable	Carrot

We can use the *searched format* of `SELECT` instead, for example:

```
SELECT
  CASE
    WHEN CategoryCode = 'F' THEN 'Fruit'
    WHEN CategoryCode = 'V' THEN 'Vegetable'
    ELSE 'Other'
  END AS 'Category',
  ProductDescription AS 'Description'
FROM Products;
```

(This returns the same result as above.)

The searched format may be useful if the returned values are determined from more than one column. If the conditions are not mutually exclusive, later `WHEN` statements override previous `WHEN` statements.

7. Row based logic

The general form of a select query is:

```
SELECT columnlist
FROM tablelist
WHERE condition
ORDER BY columnlist;
```

The keyword `WHERE` allows us to filter the rows that are returned by a query.

Let's say that we have the following table in our database:

OrderID	FirstName	LastName	QuantityPurchased	PricePerItem
1	William	Smith	4	2.50
2	Natalie	Lopez	10	1.25
3	Brenda	Harper	5	4.00

The query:

```
SELECT
  FirstName,
  LastName,
  QuantityPurchased
FROM Orders
WHERE LastName = 'Harper';
```

Returns:

FirstName	LastName	QuantityPurchased
Brenda	Harper	5

`WHERE` clause operators include `=`, `<>`, `<`, `>`, `<=`, `>=`. The operators do not only work with numerical datatypes, and can also be used in `CASE` structures.

We can set a limit to the number of rows returned using the `LIMIT` keyword (in MySQL). In this case, a query might look something like:

```
SELECT columnlist
FROM table
LIMIT number;
```

(In conjunction with `ORDER BY` this can be useful in obtaining top or bottom *n* items.)

8. Boolean logic

The keywords `AND` , `OR` , `NOT` , `BETWEEN` , `IN` , `IS` and `NULL` can be used to apply Boolean logic to queries.

A note on precedence: `AND` has a higher order of precedence than `OR` , for example, and it is often a good idea to use parentheses in complicated logic expressions to avoid bugs and aid in script reading.

Let's say that we have the following table in our database:

OrderID	CustomerName	State	QuantityPurchased	PricePerItem
1	William Smith	IL	4	2.50
2	Natalie Lopez	CA	10	1.25
3	Brenda Harper	NY	5	4.00

The query:

```
SELECT
  CustomerName,
  QuantityPurchased
FROM Orders
WHERE QuantityPurchased > 4 AND NOT State = 'NY';
```

Returns:

CustomerName	QuantityPurchased
Natalie Lopez	10

The `BETWEEN` keyword can often simplify logic expressions. For example, the query:

```
SELECT
  CustomerName,
  QuantityPurchased
FROM Orders
WHERE QuantityPurchased >= 4 AND QuantityPurchased <= 10;
```

is equivalent to:

```sql
SELECT
  CustomerName,
  QuantityPurchased
FROM Orders
WHERE QuantityPurchased BETWEEN 4 AND 10;
```

(Notice that `BETWEEN` includes its endpoints!)

Similarly the `IN` keyword can simplify logical expressions. For example, the query:

```sql
SELECT
  CustomerName,
  QuantityPurchased
FROM Orders
WHERE QuantityPurchased = 2
  OR QuantityPurchased = 5
  OR QuantityPurchased = 11;
```

is equivalent to:

```sql
SELECT
  CustomerName,
  QuantityPurchased
FROM Orders
WHERE QuantityPurchased IN (2, 5, 11);
```

The phrase `IS NULL` returns true if a value is missing.

9. Inexact matches

We can use *pattern matching* to look for inexact matches within strings.

Let's say that we have the following table in our database:

MovieID	MovieTitle
1	Love Actually
2	His Girl Friday
3	Love and Death
4	Sweet and Lowdown
5	Everyone Says I Love You
6	Down With Love
7	101 Dalmations

Now the query:

```
SELECT MovieTitle AS 'Movie'
FROM Movies
WHERE MovieTitle LIKE '%LOVE%';
```

Returns:

Movie
Love Actually
Love and Death
Everyone Says I Love You
Down With Love

Notice that `%` acts as a *wildcard* when used with `LIKE`. (Note that Oracle is case sensitive whereas MySQL and MS SQL Server are not. In Oracle we could use, for example, `WHERE UPPER(MovieTitle) LIKE '%LOVE%` to get the same result.)

Wildcards include:

Wildcard	Meaning
`%`	Any set (including empty) of characters.
`_`	Exactly one (any) character.
`[characterlist]`	Exactly one character that appears in *characterlist*.
`[^characterlist]`	Exactly one character that does not appear in *characterlist*.

The `SOUNDEX(word)` function returns a four-character code that is intended to approximately map the sound of a word. Two words that sound similar will have similar codes. This functionality may be used to search a table based on the sound of an entry.

10. Summarizing data

Eliminating duplicates

We can eliminate duplicates from a table using the `DISTINCT` keyword, placed immediately after `SELECT`.

Let's say that we have the following table in our database:

SongID	Artist	Album	Title
1	The Beatles	Abbey Road	Come Together
2	The Beatles	Abbey Road	Sun King
3	The Beatles	Revolver	Yellow Submarine
4	The Rolling Stones	Let It Bleed	
5	The Rolling Stones	Flowers	Ruby Tuesday
6	Paul McCartney	Ram	Smile Away

The query:

```
SELECT DISTINCT Artist
FROM SongTitles
ORDER BY Artist;
```

Returns:

Artist
Paul McCartney
The Beatles
The Rolling Stones

Aggregate functions

Aggregate functions are used on columns of data and are meant to provide summaries. Commonly used aggregate functions include:

- `COUNT`
- `SUM`
- `AVG`
- `MIN`
- `MAX`

Let's say that we have the following table in our database:

Grades

GradeID	Student	GradeType	Grade
1	Susan	Quiz	92
2	Susan	Quiz	95
3	Susan	Homework	84
4	Kathy	Quiz	62
5	Kathy	Quiz	81
6	Kathy	Homework	NULL
7	Alec	Quiz	58
8	Alec	Quiz	74
9	Alec	Homework	88

We can create a summary of the quiz scores using some of the agregate functions:

```sql
SELECT
  AVG(Grade) AS 'AverageQuizScore',
  MIN(Grade) AS 'MinimumQuizScore',
  MAX(Grade) AS 'MaximumQuizScore'
FROM Grades
WHERE GradeType = 'Quiz';
```

Returns:

AverageQuizScore	MinimumQuizScore	MaximumQuizScore
77	58	95

We can count the rows in the table using `COUNT` :

```sql
SELECT COUNT(*) AS rows
FROM Grades;
```

Note that `NULL` values aren't counted by `COUNT` . If a column (or coulumn list) is not specified (as in the above example) only rows that consist entirely of `NULL` values will not be counted.

Grouping data

We can use `GROUP BY` in our query to apply aggregate functions to different row categories. For example, the query:

```sql
SELECT
  GradeType AS 'GradeType',
  AVG(Grade) AS 'Mean'
FROM Grades
GROUP BY GradeType
ORDER BY GradeType;
```

Returns:

GradeType	Mean
Homework	86
Quiz	77

The `HAVING` keyword can be used similarly to the `WHERE` keyword, but on aggregated data. For example, the query:

```sql
SELECT
  Student AS 'Student',
  AVG(Grade) AS 'AverageQuizGrade'
FROM Grades
WHERE GradeType = 'Quiz'
GROUP BY Student
HAVING AVG(Grade) >= 70
ORDER BY Student
```

Produces:

Student	AverageQuizGrade
Kathy	71.5
Susan	93.5