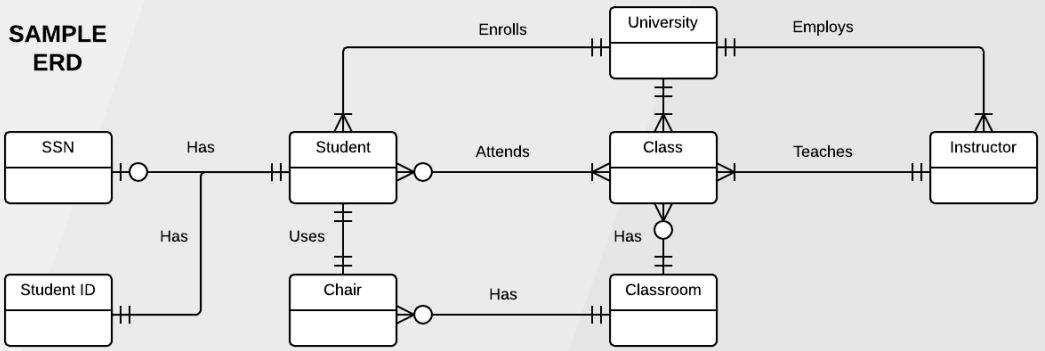
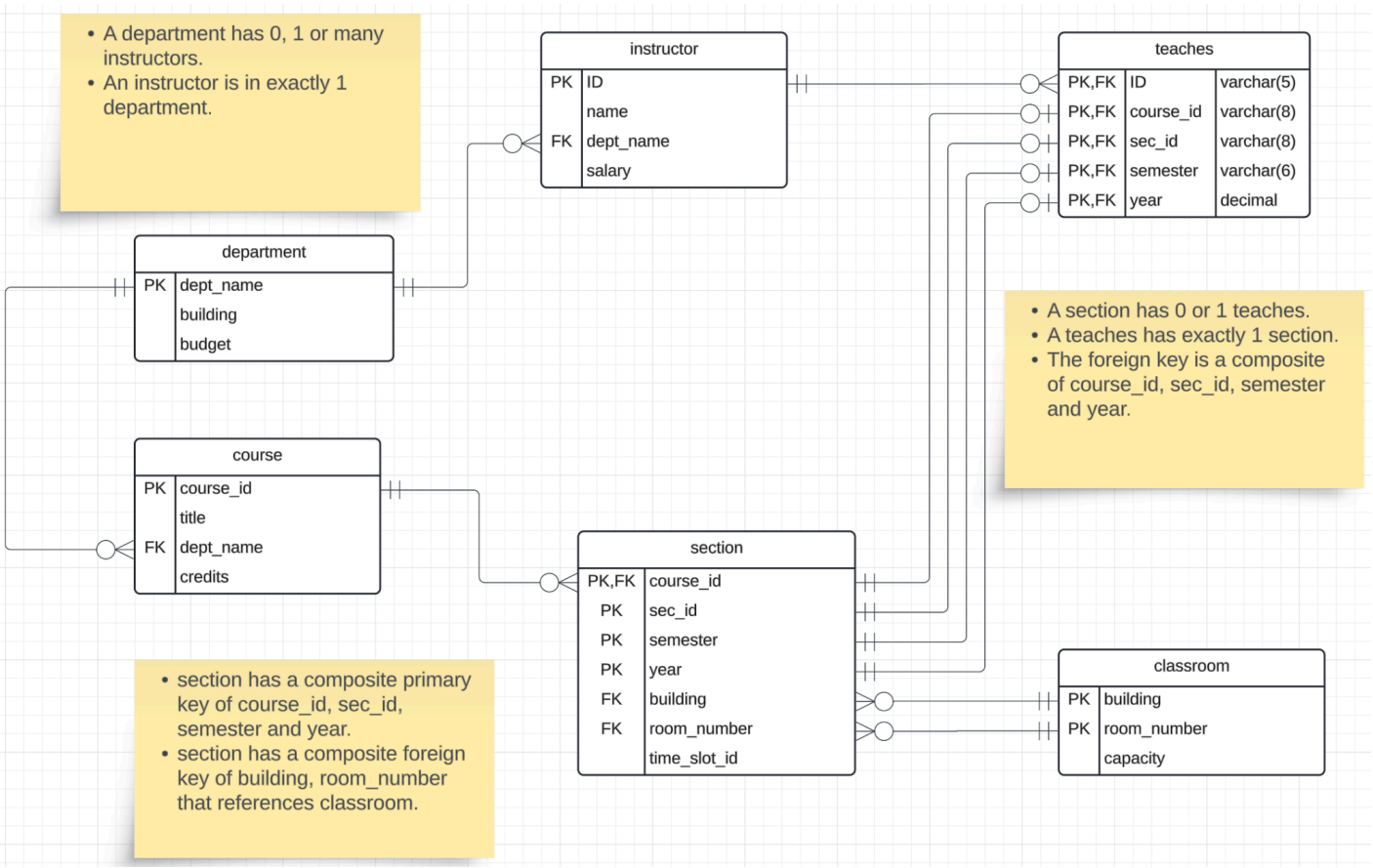
ERD "Crow's Foot" Relationship Symbols [Quick Reference]

Created by Vivek M. Chawla | @VivekMChawla | April 7, 2013

SAMPLE ERD



Notation	Meaning	Example
	Relationship	Student — Enrolls — University
	One	Student — Has — Student ID Number
	Many	Student — Attends — Class
	One and ONLY One	Student — Uses — Chair
	Zero or One	Student — Has — Social Security Number
	One or Many	Instructor — Teaches — Class
	Zero or Many	Classroom — Has — Chair


• A department has 0, 1 or many instructors.
• An instructor is in exactly 1 department.
instructor
PK ID
name
FK dept_name
salary
teaches
PK,FK ID varchar(5)
PK,FK course_id varchar(8)
PK,FK sec_id varchar(8)
PK,FK semester varchar(6)
PK,FK year decimal
department
PK dept_name
building
budget
• A section has 0 or 1 teaches.
• A teaches has exactly 1 section.
• The foreign key is a composite of course_id, sec_id, semester and year.
course
PK course_id
title
FK dept_name
credits
section
PK,FK course_id
PK sec_id
PK semester
PK year
FK building
FK room_number
time_slot_id
classroom
PK building
PK room_number
capacity
• section has a composite primary key of course_id, sec_id, semester and year.
• section has a composite foreign key of building, room_number that references classroom.

- τ order by
- γ group by
- $\neg$ negation
- $\div$ set division
- $\bowtie$ natural join, theta-join
- $\ltimes$ left outer join
- $\rtimes$ right outer join
- $\ltimes\rtimes$ full outer join
- $\ltimes$ left semi join
- $\rtimes$ right semi join
- $\triangleright$ anti-join

```
σ student_dept='Comp. Sci.' ∨ student_dept='Elec. Eng.'
(
  π student_id, student_name, student_dept,
    advisor_id, advisor_name, advisor_dept
  (
    (
      π student_id←ID, student_name←name,
        student_dept←dept_name, i_id
            (student ⋈ ID=s_id advisor)
    )
    ⋈ i_id=advisor_id
    (
      π advisor_id←ID, advisor_name←name, advisor_dept←dept_name
            (instructor ⋈ ID=i_id advisor)
    )
  )
)
```

Result

student_id	student_name	student_dept	advisor_id	advisor_name	advisor_dept
128	Zhang'	Comp. Sci.'	45565	Katz'	Comp. Sci.'
12345	Shankar'	Comp. Sci.'	10101	Srinivasan'	Comp. Sci.'
76543	Brown'	Comp. Sci.'	45565	Katz'	Comp. Sci.'
76653	Aoi'	Elec. Eng.'	98345	Kim'	Elec. Eng.'
98765	Bourikas'	Elec. Eng.'	98345	Kim'	Elec. Eng.'

SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE  c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING TABLES

```
CREATE TABLE t (
    id INT PRIMARY KEY,
    name VARCHAR NOT NULL,
    price INT DEFAULT 0
);
```
Create a new table with three columns

```
DROP TABLE t ;
```
Delete the table from the database

```
ALTER TABLE t ADD column;
```
Add a new column to the table

```
ALTER TABLE t DROP COLUMN c ;
```
Drop column c from the table

```
ALTER TABLE t ADD constraint;
```
Add a constraint

```
ALTER TABLE t DROP constraint;
```
Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```
Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2 ;
```
Rename column c1 to c2

```
TRUNCATE TABLE t;
```
Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
   c1 INT, c2 INT, c3 VARCHAR,
   PRIMARY KEY (c1,c2)
);
```
Set c1 and c2 as a primary key

```
CREATE TABLE t1(
   c1 INT PRIMARY KEY,
   c2 INT,
   FOREIGN KEY (c2) REFERENCES t2(c2)
);
```
Set c2 column as a foreign key

```
CREATE TABLE t(
   c1 INT, c1 INT,
   UNIQUE(c2,c3)
);
```
Make the values in c1 and c2 unique

```
CREATE TABLE t(
  c1 INT, c2 INT,
  CHECK(c1> 0 AND c1 >= c2)
);
```
Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 VARCHAR NOT NULL
);
```
Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
       (value_list), ....;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
    c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name;
```
Delete a view

MANAGING INDEXES

```
CREATE INDEX idx_name
ON t(c1,c2);
```
Create an index on c1 and c2 of the table t

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
```
Create a unique index on c3, c4 of the table t

```
DROP INDEX idx_name;
```
Drop an index

SQL AGGREGATE FUNCTIONS

AVG returns the average of a list

COUNT returns the number of elements of a list

SUM returns the total of a list

MAX returns the maximum value in a list

MIN returns the minimum value in a list

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```
Create or modify a trigger

WHEN
- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

EVENT
- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

TRIGGER_TYPE
- **FOR EACH ROW**
- **FOR EACH STATEMENT**

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```
Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```
Delete a specific trigger

> Definitions used throughout this cheat sheet

Primary key:
A primary key is a field in a table that uniquely identifies each record in the table. In relational databases, primary keys can be used as fields to join tables on.

Foreign key:
A foreign key is a field in a table which references the primary key of another table. In a relational database, one way to join two tables is by connecting the foreign key from one table to the primary key of another.

One-to-one relationship:
Database relationships describe the relationships between records in different tables. When a one-to-one relationship exists between two tables, a given record in one table is uniquely related to exactly one record in the other table.

One-to-many relationship:
In a one-to-many relationship, a record in one table can be related to one or more records in a second table. However, a given record in the second table will only be related to one record in the first table.

Many-to-many relationship:
In a many-to-many relationship, records in a given table 'A' can be related to one or more records in another table 'B', and records in table B can also be related to many records in table A.

> Sample Data

Artist Table

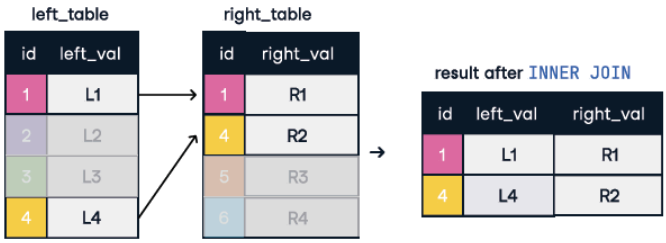
artist_id	name
1	AC/DC
2	Aerosmith
3	Alanis Morissette

Album Table

album_id	title	artist_id
1	For those who rock	1
2	Dream on	2
3	Restless and wild	2
4	Let there be rock	1
5	Rumours	6

INNER JOIN

An inner join between two tables will return only records where a joining field, such as a key, finds a match in both tables.



INNER JOIN join ON one field

```
SELECT *
FROM artist AS art
INNER JOIN album AS alb
ON art.artist_id = alb.artist_id;
```

Result after INNER JOIN:

artist_id	name	title	album_id
1	AC/DC	For those who rock	1
1	AC/DC	Let there be rock	4
2	Aerosmith	Dream on	2
2	Aerosmith	Restless and wild	3

INNER JOIN with USING

```
SELECT *
FROM artist AS art
INNER JOIN album AS alb
USING (artist_id);
```

SELF JOIN

Self-joins are used to compare values in a table to other values of the same table by joining different parts of a table together.

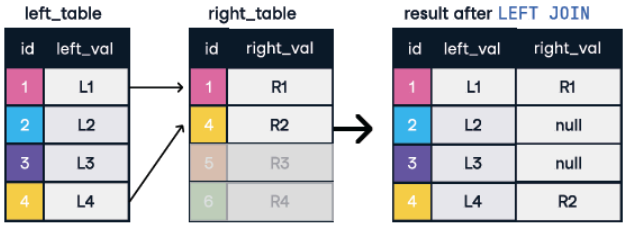
```
SELECT
  alb1.artist_id,
  alb1.title AS alb1_title,
  alb2.title AS alb2_title
FROM album AS alb1
INNER JOIN album AS alb2
ON art1.artist_id = art2.artist_id
WHERE alb1.album_id<>alb2.album_id;
```

Result after Self join:

artist_id	name	album_id	alb2_title
1	AC/DC	1	For those who rock
2	Aerosmith	2	Dream on
2	Aerosmith	3	Restless and wild
1	AC/DC	4	Let there be rock

LEFT JOIN

A left join keeps all of the original records in the left table and returns missing values for any columns from the right table where the joining field did not find a match.



LEFT JOIN on one field

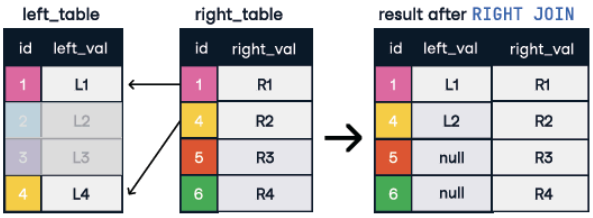
```
SELECT *
FROM artist AS art
LEFT JOIN album AS alb
ON art.artist_id = alb.artist_id;
```

Result after LEFT JOIN:

artist_id	name	album_id	title	name
1	AC/DC	1	For those who rock	1
1	AC/DC	4	Let there be rock	1
2	Aerosmith	2	Dream on	2
2	Aerosmith	3	Restless and wild	2
3	Alanis Morissette	null	null	null

RIGHT JOIN

A right join keeps all of the original records in the right table and returns missing values for any columns from the left table where the joining field did not find a match. Right joins are far less common than left joins, because right joins can always be re-written as left joins.



RIGHT JOIN on one field

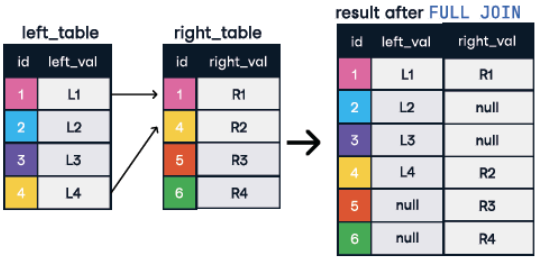
```
SELECT *
FROM artist as art
RIGHT JOIN album AS alb
ON art.artist_id = alb.artist_id;
```

Result after RIGHT JOIN:

artist_id	name	album_id	title	name
1	AC/DC	1	For those who rock	1
1	Aerosmith	2	Dream on	2
2	Aerosmith	3	Restless and wild	2
2	AC/DC	4	Let there be rock	1
3	null	5	Rumours	6

FULL JOIN

A full join combines a left join and right join. A full join will return all records from a table, irrespective of whether there is a match on the joining field in the other table, returning null values accordingly.



FULL JOIN on one field

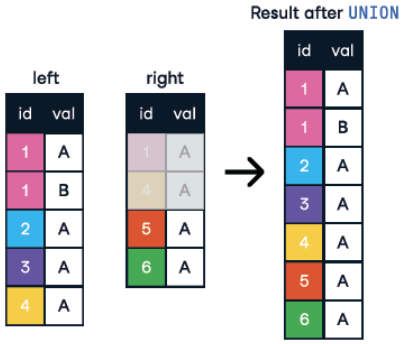
```
SELECT *
FROM artist as art
FULL OUTER JOIN album AS alb
ON art.artist_id = alb.artist_id;
```

Result after FULL JOIN:

artist_id	name	album_id	title	name
1	AC/DC	1	For those who rock	1
1	AC/DC	4	Let there be rock	1
2	Aerosmith	2	Balls to the wall	2
2	Aerosmith	3	Restless and wild	2
3	Alanis Morissette	null	null	null
null	null	5	Rumours	6

UNION

The `UNION` operator is used to vertically combine the results of two `SELECT` statements. For `UNION` to work without errors, all `SELECT` statements must have the same number of columns and corresponding columns must have the same data type. `UNION` does not return duplicates.



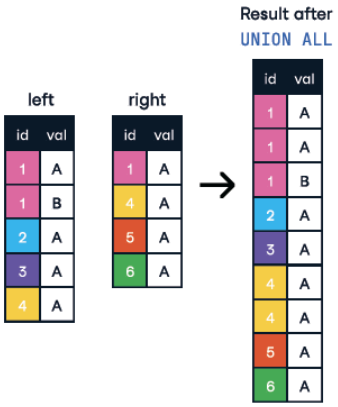

```
SELECT artist_id
FROM artist
UNION
SELECT artist_id
FROM album;
```

Result after UNION:

artist_id
1
2
3
6

UNION ALL

The `UNION ALL` operator works just like `UNION`, but it returns duplicate values. The same restrictions of `UNION` hold true for `UNION ALL`




```
SELECT artist_id
FROM artist
UNION ALL
SELECT artist_id
FROM album;
```

Result after UNION ALL:

artist_id
1
2
3
1
2
2
1
6