# Joining Data

Returning results with data from multiple tables

# Joining Data

## The Basics

- Properly normalized database:
  - Related data is in different tables
  - Simple select statements fail to merge data
- Commonalities exist between tables:
  - Linking field in the ‘Many’ table
  - ID field in the ‘One’ table

# Joining Data

## Using the JOIN Clause

- Tables are ‘Joined’ together
- Definition of how the join is made
  - Typical: one.ID_field = many.linking_field
  - Variations: other links can be defined
- Field listings may need table prefixes

# Joining Data

## Using the JOIN Clause

```
SELECT companies.name, orders.product
FROM companies
JOIN orders
ON companies.company_id = orders.company;
```

# Joining Data

## Using the JOIN Clause

Default behavior returns ALL data

- Data in the ‘One’ table
- No matching data in ‘Many’ table
- Results show ‘One’ data and null in ‘Many’
- Same is true for reverse

# Joining Data

## Using the JOIN Clause

Method of retrieving data can be specified

- Linked data must exist in both tables - INNER
- Data in one of the tables determines inclusion in the results - OUTER
  - Specify which ‘side’ of the join has precedence

# Joining Data

## Using the JOIN Clause

```
SELECT companies.name, orders.product
FROM companies
INNER JOIN orders
ON companies.company_id = orders.company;
```

Only data where a company has orders are shown

# Joining Data

## Using the JOIN Clause

```
SELECT companies.name, orders.product
FROM companies
LEFT OUTER JOIN orders
ON companies.company_id = orders.company;
```

All companies are listed, even if they don’t have any orders

# Joining Data

## Using the JOIN Clause

```
SELECT companies.name, orders.product
FROM companies
RIGHT OUTER JOIN orders
ON companies.company_id = orders.company;
```

All orders are listed, even if they don’t belong to a company

# Aliases

## Simplifying SQL Statements

- Select statements can quickly grow and become complex
- Use of table aliases help manage length and improve readability
- Keyword ‘AS’ is used to designate alias

# Aliases

## Simplifying SQL Statements

```
SELECT c.name, o.product
FROM companies AS c
LEFT OUTER JOIN orders AS o
ON c.company_id = o.company;
```

# Foreign Keys

Establish dependencies between records
in different tables

# Foreign Keys

## Definition

- A link between two fields of different tables
  - Used to associate a related record to the master record
  - An index is normally built to manage the link
- The ‘Key’ that is used is based upon a foreign (from another table) value

# Foreign Keys

## Usage

- Establishes the relationship between tables

**parent.field = child.field**

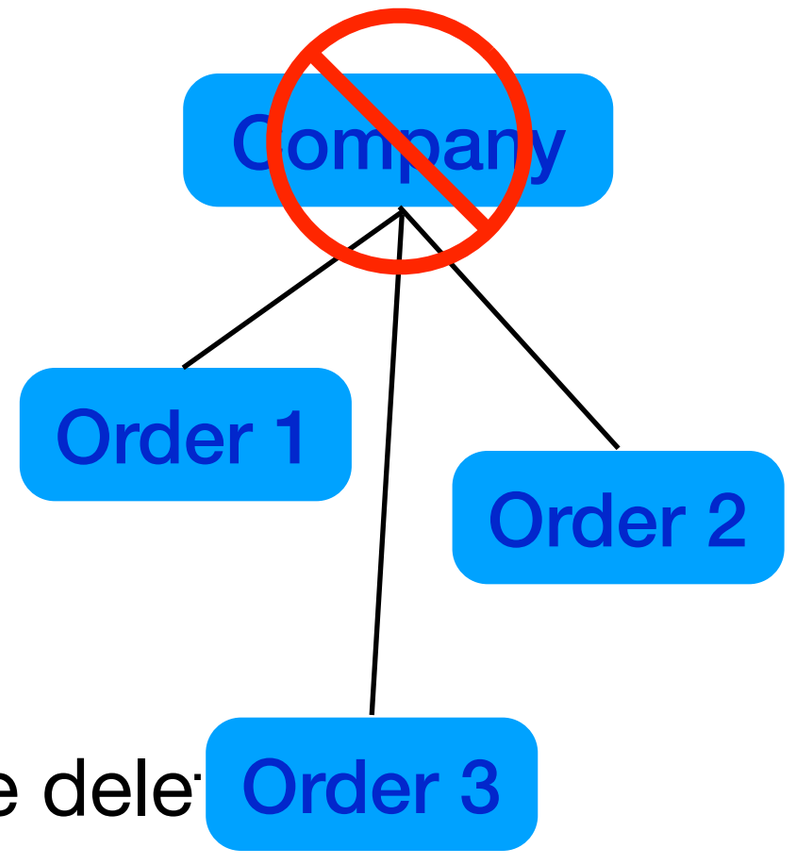
- Mechanism for enforcing referential integrity rules

# Foreign Keys

## Referential Integrity

Integrity Rule Options:

- Cascade Deletion
  - Parent record is deleted
  - All related children records are dele




DANGER

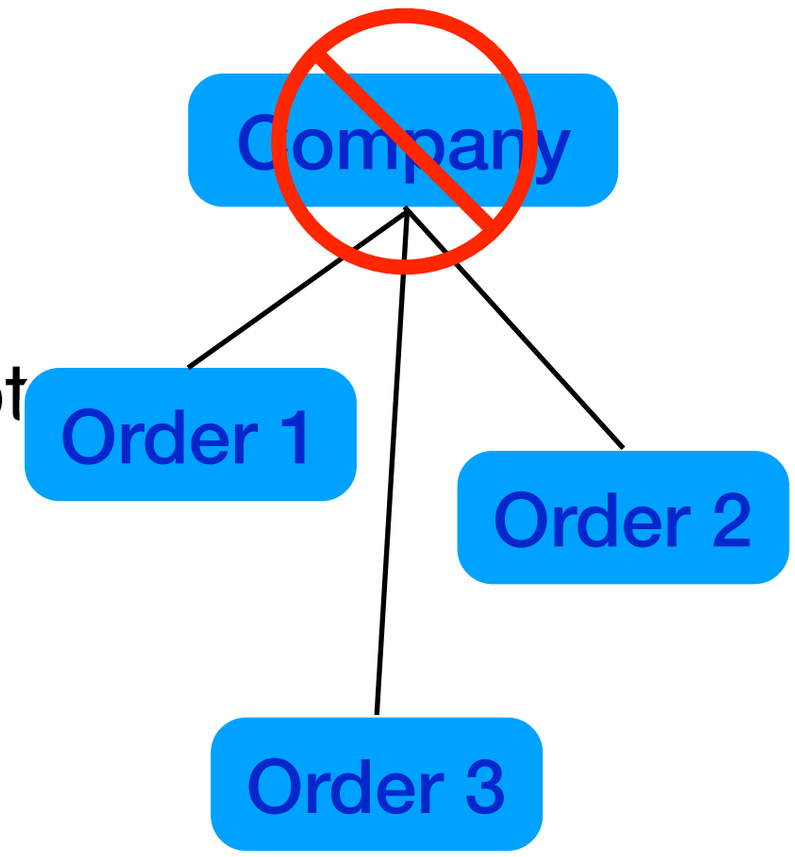
**No warning message with this setup**

# Foreign Keys

## Referential Integrity

Integrity Rule Options:

- Restrict Deletion
- Parent record deletion attempt
- Child records exist
- Error message is returned
- No deletion made



# Foreign Keys

## Creating in SQL

- Use the `ALTER TABLE` command and...
- Use the `ADD FOREIGN KEY` clause

**OR**

- Include in the `CREATE TABLE` command
- Each option requires the `REFERENCES` keyword to complete the setup

# Foreign Keys

## Creating in SQL

```
ALTER TABLE <related table>
ADD FOREIGN KEY <linking field>
REFERENCES <parent table> (<unique field>);
```

```
ALTER TABLE invoices
ADD FOREIGN KEY customer
REFERENCES customers (customer_id);
```

# Foreign Keys

## Creating in SQL

```
ALTER TABLE invoices
ADD FOREIGN KEY customer
REFERENCES customers (customer_id)
ON DELETE CASCADE;
```

```
ALTER TABLE invoices
ADD FOREIGN KEY customer
REFERENCES customers (customer_id)
ON DELETE RESTRICT;
```

# Foreign Keys

## Creating in SQL

```
CREATE TABLE invoices {
    invoice_id integer,
    customer_number int REFERENCES customers (customer_number),
    total decimal,
    taxes decimal,
    shipping decimal,
    invoice_date date
};
```

# Foreign Keys

## Creating in SQL

```
CREATE TABLE invoices {
    invoice_id integer,
    customer_number int
            REFERENCES customers (customer_number)
            ON DELETE CASCADE,
    total decimal,
    taxes decimal,
    shipping decimal,
    invoice_date date
};
```

# Foreign Keys

## Creating in SQL

```
CREATE TABLE invoices {
    invoice_id integer,
    customer_number int
            REFERENCES customers (customer_number)
            ON DELETE RESTRICT,
    total decimal,
    taxes decimal,
    shipping decimal,
    invoice_date date
};
```